Proofs of partial correctness for attribute grammars and recursive procedures

Bruno Courcelle, Pierre Deransart

IRIA
CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P. 105
78153 Le Chesnay Cedex
France
Tél. : (1) 39 63 55 11

Rapports de Recherche

N° 322

PROOFS OF PARTIAL CORRECTNESS FOR ATTRIBUTE GRAMMARS AND RECURSIVE PROCEDURES

Bruno COURCELLE
Pierre DERANSART

Juillet 1984

Proofs of partial correctness for attribute grammars and recursive procedures

By Bruno COURCELLE[+]
Pierre DERANSART[*]

[+] Bordeaux-I University
Dept. of Mathematics
33405 TALENCE, FRANCE

[*] INRIA
Domaine de Voluceau-Rocquencourt
B.P.105
78153 LE CHESNAY CEDEX, FRANCE

Abstract : An extension of the inductive assertion method allowing to prove the partial correctness of an attribute grammar w.r.t. a specification is presented. It is complete in an abstract sense. It is also shown that the semantics of systems of recursive imperative procedures or of recursive applicative procedures computed with call-by-value or call-by-name can be expressed by an attribute grammar associating attributes to the nodes of the so-called trees of calls. Hence the proof methods for the partial correctness of attribute grammars can be applied to these recursive procedures.

Note : This work has been supported by the "GRECO de programmation" (ATTRISEM project).

Ce travail a été réalisé dans le cadre du "GRECO de programmation" (projet ATTRISEM)

Résumé : Preuves de validité partielle des grammaires d'attributs et des procédures récursives.

On présente dans ce rapport une extension de la méthode des assertions inductives utilisée pour prouver la validité partielle des grammaires d'attributs par rapport à une spécification donnée. La méthode est complète en un sens abstrait. On montre également que la sémantique des systèmes de procédures récursives impératives peut-être exprimée par une grammaire d'attributs qui associe les attributs aux noeuds d'un arbre d'appels. Il en est de même pour les procédures récursives applicatives avec appel par valeur et appel par nom. La méthode de preuve de correction partielle s'applique de fait à ces procédures.

Mots clé : Méthode des assertions, validité partielle, grammaires d'attributs, procédures récursives, arbre d'appels.

(0) Introduction

The problem of proving the validity of an attribute grammar with respect to a given specification, saying what the values of the attributes should be, although essential, has been rarely considered [ANP79, KH81, Der83, Cou83].

Since an attribute grammar, say for specifying a compiler, can be very large, proving its validity, i.e. the correctness of the generated compiler is clearly not a trivial task.

We only consider here the partial correctness of an attribute grammar with respect to a specification. A specification consists in a logical formula associated with each non-terminal and establishing a relation between the values of the attributes at a node labelled by this non-terminal. An attribute grammar is partially correct if, for all tree t, for all assignment of values at the attributes at the nodes of the tree which satisfies the semantic rules, then the specification is satisfied at the root of the tree.

In terms of flowcharts, this corresponds to saying that for all computation paths, the output values satisfy some prescribed relation with the input ones.

The total correctness of an attribute grammar would consist in requiring the existence of at least one tree, with at least one assignment of values to attributes satisfying some condition. This problem will not be considered here (it is in [Der84]).

We propose two proof methods for establishing the partial correctness of an attribute grammar. The first one consists in defining a specification stronger than the original one, which furthermore is inductive. This means that for each node of a derivation tree, the validity of the specification at this node follows from the validity of the specification at its sons and the semantic rules of the production associated with this node.

Since the strongest specification is inductive this proof method is complete, provided one accepts for specifications arbitrary set theoretical relations as opposed to, say, first-order formulas. Otherwise, one gets an incompleteness result similar to Wand's for the inductive assertion method [Wan78]

Our second method is a refinement of the first one. We associate with every non-terminal a finite set of formulas (we call this annotation), together with implications between formulas (and using the semantic rules as premi es) some formulas are called inherited, other ones synthesized. They behave with respect to implications as inherited and synthesized attributes behave w.r.t. dependencies, and we require non-circularity exactly as we do for ordinary attribute grammars. This non-circularity insures the non-circularity of the proof that, for all tree, if the inherited formulas are valid at the root then so are the synthesized ones.

In order to prove the validity of a specification θ, it suffices to find an annotation which is non-circular and such that, for all non-terminal :

$$\left(\left(\mathcal{M}_{\mathcal{I}} \Rightarrow \mathcal{M}_{\mathcal{S}} \right) \Rightarrow \theta \right)$$

where $\mathcal{I}$ (resp. $\mathcal{S}$.) is the set of inherited (resp. synthesized) formulas at the root.

The first method corresponds to taking one synthesized formula for each non-terminal (and the non-circularity is then trivial as for purely synthesized attribute grammars). The second one consists in decomposing specification into implications between conjucts of simpler formulas ; it is certainly useful in practice to get clearer proofs.

We also think that the structure of implications can follow the structure of dependencies between the attributes of the attribute grammar to be validated.

Note that our proof methods do not rest at all on the non-circularity of the given attribute grammar as did the ones of [PAN79] or [KH81]. (Recall that the partial correctness is stated as "for all tree, for all assignment satisfying the semantic rules..."; there may exist none or many for a circular attribute grammar). Hence we formulate our methods for a generalization of attribute grammars

called relational attribute grammars where the semantic rules do not state that some attribute is a function of other ones but simply state relations (possibly not functional in any way) between attributes. Hence, there is no distinction between synthesized and inherited attributes. Clearly the non-circularity is meaningless for relational attribute grammars.

In our proof method, we need a non-circularity at the level of logical formulas, not at the level of the attribute grammar we are validating.

The second part of this paper is devoted to the characterization of the semantics of recursive procedures of various kinds (imperative as in Gallier [Gal 81] applicative and evaluated by call-by-value or call-by-name as in Vuillemin [Vui 74]) in terms of a relational attribute grammar.

This relational attribute grammar is based on the set of trees of calls of a recursive procedure. Each node corresponds to "a call" (the root corresponds to the "main program"), is labeled by the name of the procedure which is called and attributes represent the parameters : the inherited attributes represent the input parameters and the synthesized attributes the ouput parameters. Each "production rule" corresponds to some alternative in the procedure body which is selected by some Boolean test. The semantic rules of such a "production" include the test together with operations performed on the program variables (which are represented by attributes).

The relational attribute grammar is of a rather simple kind : it is an L-attribute grammar (evaluable in one left-to-right pass [EF81]) augmented with Boolean conditions.

This attribute grammar is used in an unusual way : the tree is not given a priori then decorated and evaluated, but constructed in interaction with the computation of attributes; this construction is oriented by the Boolean conditions which determine it completely if the procedures are deterministic or only forbid some alternatives if they are non-deterministic.

By means of this construction the partial correctness of recursive procedures can be expressed as the partial correctness of the associated attribute grammar, and the proof methods for attribute grammars are applicable. They yield another proof of Gallier's completeness results for the provability of the $\forall$-partial

correctness of recursive imperative procedures [Gal81] and new results for the partial correctness of applicative ones.

(1) Basic definitions and notations

(1.1) Sorts, Signatures, Terms.

Let S be a finite set of sorts. An S-sorted signature (or simply an S-signature) is a finite set P of function symbols given with two mappings :

$\alpha : P \longrightarrow S^*$ ($\alpha(P)$ is called the arity of p in P)

$\sigma : P \longrightarrow S$ ($\sigma(P)$ is called the sort of p in P)

The length of $\alpha(p)$ is called the rank of p and is denoted by $\rho(p)$. If $\alpha(p) = \varepsilon$ (we denote by ε the empty word of any free monoïd) then p is a constant symbol. The pair $\langle \alpha(p), \sigma(p) \rangle$ is the profile of p. A constant or a variable has profile $\langle \varepsilon, s \rangle$. $P_{\langle w,s \rangle}$ denotes the subset of P consisting of functions of profile $\langle w,s \rangle$.

A heterogeneous P-Magma (or P-Algebra) is an objet $\mathbb{M}$:

$$\mathbb{M} = \langle \{M_s\}_{s \in S}, \{P_M\}_{p \in P} \rangle$$

where $\{M_s\}$ is a family of sets indexed by S, the carriers and each p_M a mapping :

$$M_{s_1} \times \ldots \times M_{s_n} \longrightarrow M_s \text{ if } p \in P_{\langle s_1 \ldots s_n, s \rangle}.$$

Unless otherwise specified, all magmas considered in the sequel will be heterogeneous.

Let X be an S-sorted set of variables (i.e. each x in X has arity ε and a sort $\sigma(x)$ in S); one can also define the free-P-magma generated by X, denoted by M(P,X). It is identified as usual with a set, of terms, "well typed" with respect to sorts and arities. Terms will be written with commas and parentheses if infix notation is not used. They will also be identified with trees in a well-known manner. The words "tree" and "term" will be synonymous in this paper and will refer to elements of some free magma. We shall denote by $M(P,X)_s$ the carrier of sort s of M(P,X), and by $M(P)_s$ the set of all terms without variables (ground terms) of sort s. M(P) is the set of all ground terms.

A term t in $M(P)_s$ is thought as denoting a value t_M in M_s for a P-Magma M.

Similarly a term t in $M(P, \{x_1,\ldots x_k\})_s$ is thought as denoting a function : $M_{\sigma(x_1)}\ldots x\ldots x\ M_{\sigma(x_k)} \longrightarrow M_s$ called a <u>derived operator</u>.

For any P-magma $\mathbb{M}$ and a S-sorted set X of variables, an <u>assignment</u> of values in M_s to variables X_s, for all s in S, is an S-indexed family of functions

$$\nu : \{\nu_s : X_s \longrightarrow M_s\}_{s\in S}$$

It is well known that this assignment can be extended in to a unique homomorphism $\hat{\nu} : M(P,X) \longrightarrow \mathbb{M}$ such that $\hat{\nu}(x) = \nu(x)$ for all x in X.

Let $\mathcal{Q}$ be an S-indexed family of propositions $\{Q_s\}_{s\in S}$, where Q_s is a unary proposition on $M(P)_s$. We say that the proposition

(1) $\forall s \in S,\ \forall t \in M(P)_s,\ Q_s(t)$

is <u>provable by structural induction</u> iff propositions (2) and (3) below are both provable :

(2) $\forall s \in S,\ \forall p \in P_{<\varepsilon,s>},\ Q_s(p)$

(3) $\forall s_1,\ldots,s_n,\ s \in S,\ \forall p \in P_{<s_1\ldots s_n,s>}\ \forall t_1 \in M(P)_{s_1}\ldots\forall t_n \in M(P)_{s_n}$:

$$Q_{s_1}(t_1)\ \underline{\text{and}}\ \ldots\ \underline{\text{and}}\ Q_{s_n}(t_n) \Longrightarrow Q_s(p(t_1,\ldots,t_n))$$

It is easy to show that a proposition is true if it is provable by structural induction (an elegant proof is given in [CM79])

(1.2) Trees and grammars

Let P be a fixed S-signature. By tree we mean an element of M(P,X). Let n_P = Max $\{\rho(p)/p \in P\}$.

For any integer n, we let [n] denote the set {1,2,..,n} if n≥1 and ∅ if n = 0.

The nodes of a tree t will be represented in Dewey notation by words in $[n_P]^*$. Whereas we shall use ε to denote the empty word of any free monoid, we shall use 0 for the empty word of the free monoid $[n_P]^*$, just to simplify some notations. We denote by Node(t) the set of nodes of a tree t; hence Node(t) $\subseteq [n_P]^*$ and $0 \in$ Node(t). It denotes the root of the tree t.

Every node u in t is an occurrence of a unique element p of P denoted by $lab_t(u)$. We define the sort of u (and denote it by $\sigma(u)^{(*)}$) as the sort of $lab_t(u)$ and the sort of t (denoted by σ(t)) : as the sort $\sigma_t(0)$ of its root.

Let u be a node of t. We denote by t/u the subtree of t issued from u.

We denote by $t[t_1/x_1,\ldots,t_k/x_k]$ the result of the simultaneous substitution of t_1 for x_1,...,t_k for x_k, where $x_1,\ldots,x_k$ are pairwise distinct variables.

If $Q \subseteq P$ and $T \subseteq M(P,X)$ we denote by Q(T) the set of trees in M(P,X) of the form $q(t_1,\ldots,t_k)$ with $q \in Q$ and $t_1,\ldots,t_k \in T$. We denote by M(P,T) the set of trees of theform $t[t_1/x_1,\ldots,t_k/x_k]$ for t in $M(P,\{x_1,\ldots x_k\})$ and $t_1,\ldots,t_k \in T$.

It is now well established that an attribute grammar associates a meaning to a derivation tree (i.e. an abstract syntax tree) rather than to a word, and this approach eliminates the need for a non-ambiguous grammar.

In order to emphasize this fact we redefine as follows a context-free grammar.

A context-free grammar is a triple <N,P,C> where N is a finite set (the "non-terminal alphabet"), P is a finite N-signature, C is a mapping associating with every p in P of profile $\langle X_1 \ldots X_n, X_0\rangle$ an (n+1)-tuple of words on some finite alphabet T (the "terminal alphabet").

(*) or $\sigma_t(u)$ if we want to emphasize the tree t of which u is a node.

The mapping C defines the concrete syntax of G as follows : the usual "production rule" associated with p as above would be $X_0 \longrightarrow w_0X_1w_1X_2\ldots X_nw_n$ where $C(p) = (w_0,w_1,w_2,\ldots,w_n)$.

All the theory of attribute grammar only depends on the part <N,P> of such a context-free grammar. Such a pair will be called an abstract context-free grammar.

Remark : *It may happen that* $M(P)_s$ *is empty even if* P *is not empty : take for instance* $P = \{p\}$ *with* $\alpha(p) = s$ *and* $\sigma(p) = s$

When considering an abstract context-free grammar <N,P>, *we shall always assume that* $M(P)_X \neq \emptyset$ *for all* X *in* N, *i.e. that the grammar is "reduced".*

(1.3) Logical languages

Our definitions will be given with respect to unspecified logical languages, so as to be as general as possible.

Here are some definitions and notations concerning logical languages in general.

A logical language is a set of formulas $\mathscr{L}$ built from a sorted set $\mathscr{V}$ of variables a (possibly empty) sorted set $\mathscr{R}$ of predicate symbols, a (possibly empty) sorted set $\mathscr{F}$ of funtion symbols (i.e. a signature), (in the special case where there is only one sort, these sets are unsorted), together with logical symbols like $\forall$, $\exists$, and, $\Rightarrow$.

If $\mathscr{W} \subseteq \mathscr{V}$ we denote by $\mathscr{L}(\mathscr{W})$ the set of formulas in $\mathscr{L}$ having their free variables in $\mathscr{W}$.

Together with $\mathscr{L}$ a class of $\mathscr{L}$-interpretations is defined which depends on S (the set of sorts), and the sets $\mathscr{R}$ and $\mathscr{F}$. An $\mathscr{L}$-interpretation $\mathbb{D}$ consists at least of a family of domains $D = \{D_s\}_{s\in S}$ together with functions and/or relations interpreting the symbols of $\mathscr{F} \cup \mathscr{R}$.

An assignment is a mapping $\nu : \mathscr{V} \longrightarrow D$ which preserves the sorts.

The definition of a language $\mathcal{L}$ also includes that of a notion of validity, associating a truth-value (in {true, false}) to ϕ, $\mathbb{D}$, ν where $\phi \in \mathcal{L}$, $\mathbb{D}$ is an $\mathcal{L}$-interpretation and ν an assignment. As usual the truth of ϕ for $(\mathbb{D}, \nu)$ will be denoted by

$$(\mathbb{D}, \nu) \models \phi$$

One assumes that

$$(\mathbb{D}, \nu) \models \phi \text{ iff } (\mathbb{D}, \nu') \models \phi$$

if ν and ν' are two assignments which coincide on the set of free variables of ϕ.

The notation $\mathbb{D} \models \phi$ means that $(\mathbb{D}, \nu) \models \phi$ for all assigment ν.

The logical connectors we shall use are and, or, not, $\Rightarrow$, $\Leftrightarrow$. For a finite set of formulas A, we denote by $\bigwedge A$ (resp. $\bigvee A$) the conjunction (resp. the disjunction) of A (with $\bigwedge \emptyset$ = true and $\bigvee \emptyset$ = false). Finally, if $\phi \in \mathcal{L}(\{v_1, \ldots, v_k\})$, we denote by $\phi[w_1/v_1, \ldots w_k/v_k]$ (or $\phi[w_i/v_i;\ 1 \leq i \leq k]$) the result of the substitution of w_i for each free occurrence of v_i (some renaming of variables may be necessary)

We give a few examples which will be used in the sequel, all defined w.r.t. a set of sorts S, and a sorted set of variables $\mathcal{V}$).

(1.3.1) Example : $\mathcal{L} = \mathcal{L}_0(\mathcal{R}, \mathcal{V})$

$\mathcal{R}$ is a finite set of predicate symbols each of them has an arity in S^+.

An atomic formula is an expression of the form $R(v_1 \ldots v_l)$ *where* $R \in \mathcal{R}$, $v_1 \ldots v_l \in \mathcal{V}$, $\alpha(R) = \sigma(v_1)\ \sigma(v_2) \ldots \sigma(v_l)$

A formula of $\mathcal{L}$ is a finite Boolean combination of atomic formulas

An $\mathcal{L}$-interpretation $\mathbb{D}$ *is a family of domains* $D = (D_s)_{s \in S}$ *equipped with relations* $R_{\mathbb{D}} \subseteq D_{s_1} \times \ldots \times D_{s_k}$ *for* $R \in \mathcal{R}$ *(with* $\alpha(R) = s_1 \ldots s_k$).

The semantical validity is standard.

(1.3.2) Example $\mathcal{L} = \mathcal{L}_1(\mathcal{R}, \mathcal{F}, \mathcal{V})$

$\mathcal{R}$ is as above, $\mathcal{F}$ is a finite S-signature. An atomic formula is an expression of the form

$R(t_1, \ldots t_k)$ *or*

not $(R(t_1, \ldots, t_k))$ *or*

$v = t_1$

where $t_1, \ldots t_k \in M(\mathcal{F}, \mathcal{V})$ $\alpha(R) = \sigma(t_1)) \ldots \sigma(t_k)$ *and* $\sigma(v) = \sigma(t_1)$.

A formula of $\mathcal{L}$ is a finite conjuction of atomic formulas.

A $\mathcal{L}$-interpretation is a relational structure as above with an additional structure of $\mathcal{F}$-magma.

The semantical validity is standard

(1.3.3) Example $\mathcal{L} = \mathcal{F}_0(\mathcal{R}, \mathcal{F}, \mathcal{V})$

$\mathcal{R}$ and $\mathcal{F}$ are as above and the = can be used. $\mathcal{L}$ is the set of first-order formulas (with sorts) written with $\mathcal{V}, \mathcal{F}, \mathcal{R}$ and the equality.

(1.4) Relational attribute grammars

(1.4.1) Definition

A relational attribute grammar is a 4-tuple $G = \langle N, P, \Phi, \mathbb{D} \rangle$ *consisting of*

(1) an abstract context free grammar $\langle N, P \rangle$

(2) a relational attribute system Φ *consisting of the following items :*

(i) a finite set $\underline{Attr}$ *of attributes such that* $\underline{Attr} = \bigcup_{X \in N} \underline{Attr}(X)$ *where* $\underline{Attr}(X)$ *is the set of attributes of* X *in* N *(one can have* $\underline{Attr}(X) \cap \underline{Attr}(X') \neq \emptyset, X \neq X'$*)*

(ii) every attribute a *in* $\underline{Attr}$ *has a sort* $\sigma(a)$ *in some set of sorts* S,

(iii) for each production p *of profile* $\langle X_1 \ldots X_n, X_0 \rangle$ *a formula* Φ_p *belonging to some logical calculus* $\mathcal{L}$ *with free variables in a set* $W(p)$ *that we now define.*

For each i *in* $\{0, \ldots, n\}$, *each* a *in* $\underline{Attr}(X_i)$ *we introduce a new symbol* $a(i)$ *and call it an* occurrence of the attribute *a in p. We denote by* $W(p)$ *the set of these symbols. We also say that* $W(p)$ *is the set of* attribute occurrences *of* p.

The sort $\sigma(a(i))$ *is defined as* $\sigma(a)$.

(3) $\mathbb{D}$ *is an* $\mathcal{L}$*-interpretation.*

Convention : The sets $\underline{Attr}(X)$ will be ordered in a fixed way so that $\underline{Attr}(X)$ will be used as a sequence (an element of $\underline{Attr}^*$) in some cases

(1.4.2) Semantics of a relational attribute grammar

Let $t \in M(P)$

We define a set of variables $W(t)$ called the set of attribute occurrences of t as

$$W(t) = \{a(u) \;/\; u \in \underline{Node}(t),\ a \in \underline{Attr}\ (\sigma_t(u))\}$$

The sort of a(u) is defined as $\sigma(a(u)) = \sigma(a)$. We shall denote by $W_0(t)$ the set $\{a(0) \;/\; a \in \underline{Attr}\ (\sigma_t(0))\}$.

For each $u \in \underline{Node}\ (t)$ we denote by Φ_u the formula

$$\Phi_p[a(ui)/a(i)\ ;\ a(i) \in W(p)]$$

Where $p = \underline{lab}_t(u)$. Recall that 0 denotes the empty word so that $uv = v$ for $u = 0$.

We denote by Φ_t the conjunction of all the Φ_u's, for $u \in Node(t)$.

An assignment $\nu : W(t) \longrightarrow D$ is called a t-assignment. It is valid if

$$(\mathbb{D}, \nu) \models \Phi_t.$$

One may be interested in all valid t-assignments or simply in their projections on the attribute occurrences at the root of t i.e. $\{a(u) \in W(t)/u = 0\} = W_0(t)$. In this case we shall use the relation $R_{t,\mathbb{D}} \subseteq D_{\sigma(a_1)} \times \ldots \times D_{\sigma(a_k)}$ (where $(a_1,\ldots,a_k) = \underline{Attr}\ (\sigma(\underline{lab}_t(0)))$)

defined by :

$(d_1,\ldots,d_k) \in R_{t,\mathbb{D}}$ iff there exists a valid t-assignment ν such that $\nu(a_i(0)) = d_i$ for $i = 1,\ldots,k$.

Note that with these definitions there is no distinction between synthesized and inherited attributes and $R_{t,\mathbb{D}}$ is always defined (but possibly empty) without any extra-condition like non-circularity.

By introducing a partition of Attr into Syn $\cup$ Inh (synthesized and inherited attributes) and formulas Φ_p in $\mathscr{L}_1(\emptyset, \mathscr{F}, W(p))$ together with some further syntactical restrictions, one obtains the usual attribute grammars as a special case. See (1.5.7) below.

(1.5) Attribute dependency schemes.

After having introduced attribute grammars without dependencies, we now introduce attribute grammars with pure dependencies between attributes and no

semantics. These new formal objects are called attribute dependency schemes.

Let us begin with some definitions and notations concerning binary relations and graphs.

(1.5.1) Relations and graphs

By a graph we mean a finite oriented simple graph formally defined as a pair $D = (A,R)$ consisting of a finite set of vertices A and a binary relation $R \subseteq A \times A$

If $(a,b) \in R$ this means that there exists an arc form a to b.

When A is known from the context we shall identify D with R(and write D for (A,D)).

We denote by D^+ the graph (A,R^+) where R^+ is the transitive closure of R.

If $A' \subseteq A$ we denote by $D \restriction A'$ the graph $D' = (A', R \cap (A' \times A'))$.

By the union of a family of graphs $(D_i)_{i \in I}$ $D_i = (A_i, R_i)$ we mean the graph

$D = (\cup\{A_i / i \in I\}, \cup\{R_i / i \in I)\})$

(1.5.2) Definition

An attribute dependency scheme is a 4-tuple $S = \langle N,P,Attr,D \rangle$

such that :

(i) $\langle N,P \rangle$ *is an abstract context-free grammar,*

(ii) Attr *is a finite set of attributes as in def.(1.4.1) (but sorts are irrelevant here)*

(iii) D *is a mapping associating with every* p *in* P *a binary relation* $D(p) \subseteq W(p) \times W(p)$ *(where* W(p) *is as in(1.4.1)) also considered as a graph with*

set of vertices W(p). *This graph will also be denoted by* D(p).

We consider D(p) as a local dependency graph. We now define D(t), the global dependency graph of some t in M(P), built by "pasting" together the D(p)'s at all nodes of t.

(1.5.3) Definition : Dependency graphs

Let $t \in M(p)$.

We denote by D(t) the graph with set of vertices W(t) (same notation as in (1.4)) and call it the dependency graph of t :

$$D(t) = \cup\{w.\ D(p)/w \in \underline{Node}(t),\ p = \underline{lab}_t(w)\}$$

$$w.\ D(p) = (w.\ W(p), \xrightarrow[w,p]{})$$

$$w.W(p) = \{\ a(wu)/a(u) \in W(p)\}$$

$$a(wu) \xrightarrow[w,p]{} b(wu') \text{ iff } (a(u),\ b(u')) \in D(p).$$

We say that S is non-circular if, for all t in M(p), D(t) has no cycle (and circular otherwise).

Deciding whether S is non-circular is a trivial extension of the usual non-circularity test for attribute grammars. See [DJL83] for practical methods and references on circularity tests.

The study of the D(t)'s necessitates some further notations.

(1.5.4) Notations

We denote by $D^+(t)$ the transitive closure of D(t), by $D_0^+(t)$ the restriction of $D^+(t)$ to $W_0(t)$ (= $\{a(u) \in W(t)/u = 0\}$). Since $W_0(t)$ is in bijection with $\underline{Attr}(\sigma_t(0))$ (by $a(0) \longmapsto a$) we shall consider $D_0^+(t)$ as a binary relation on $\underline{Attr}(\sigma_t(0))$ i.e. the attributes of the root of t.

Let $p \in P_{<X_1 \ldots X_n, X_0>}$.

Let $r_i \subseteq \underline{Attr}(X_i) \times \underline{Attr}(X_i)$ for $i = 1,\ldots n$.

We denote by $D(p)\ [r_1,\ldots,r_n]$ the graph $D(p) \cup 1.r_1 \cup \ldots \cup n.r_n$ where $i.r_i$ is the graph with set of vertices $\{a(i)/a \in \underline{Attr}(X_i)\}$ and set of arcs $\{(a(i), b(i))/(a,b) \in r_i\}$

We denote by $D_0^+(p)\ [r_1\ldots,r_{n_p}]$ the restriction to $W_0(p) = \{a(0)/a \in \underline{Attr}(X_0)\}$ of $(D(p)[r_1,\ldots,r_{n_p}])^+$. Through the bijection $a \longrightarrow a(0)$ of $\underline{Attr}(X_0) \longrightarrow W_0(p)$ we consider $D_0^+(p)\ [r_1,..,r_{n_p}]$ as a binary relation on $\underline{Attr}(X_0)$.

(1.5.5) Definition

We now introduce a mapping $\mathscr{D}$ *such that* $\mathscr{D}(X)$ *represents (or approximates in some sense) the set of* $D_0^+(t)$*'s for all* t *in* $M(P)_X$ *i.e. the set of possible dependencies at the root of trees in* $M(P)_X$.

Let $\mathscr{D}$ *be a mapping associating with every* X *in* N *a finite subset* $\mathscr{D}(X)$ *of transitive relations on* $\underline{Attr}(X)$.

Such a mapping is called a dependency indicator.

We say that $\mathscr{D}$ *is* non-circular *iff, for all* p *in* $P_{<X_1\ldots X_n,X_0>}$, *all* r_1 *in* $\mathscr{D}(X_1)$ *,...,*r_n *in* $\mathscr{D}(X_n)$, $D(p)\ [r_1,\ldots,r_n]$ *is non-circular.*

It is closed *if for such* p, $r_1,\ldots,r_n$ *there exists* r_0 *in* $\mathscr{D}(X_0)$ *such that*

$$D_0^+(p)\ [r_1,\ldots,r_{n_p}] \subseteq r_0.$$

We shall compare any two dependency indicators $\mathscr{D}$ and $\mathscr{D}'$ by letting : $\mathscr{D} \leq \mathscr{D}'$ iff for every X in N, r in $\mathscr{D}(X)$ there exists r' in $\mathscr{D}'(X)$ such that $r \subseteq r'$.

Let us denote by $\mathscr{D}_0$ the dependency indicator such that $\mathscr{D}_0(X) = \{D_0^+(t)/ t \in M(P)_X\}$.

(1.5.6) Proposition :

1) $\mathscr{D}_0$ *is closed. If* $\mathscr{D}$ *is any closed dependency indicator, then* $\mathscr{D}_0 \leq \mathscr{D}$.

2) S *is non-circular iff* $\mathscr{D}_0$ *is non-circular iff there exists a closed and non-circular dependency indicator for* S.

Proof : 1) That $\mathcal{D}_0$ is closed is an easy consequence of the definition.

It is easy to prove by induction on the structure of t that, for all t in M(P), there exists r in $\underline{\mathcal{D}}(\sigma(t))$ such than $D_0^+(t) \subseteq r$.

2) Remark first that S is circular iff there exists p, $t_1,\ldots,t_n$ such that $D(p)\ [D_0^+(t_1),\ldots,D_0^+(t_n)]$ has cycles. Hence S is non-circular iff $\mathcal{D}_0$ is iff there exist for S some non-circular and closed dependency indicator, since if $\mathcal{D}$ is such than $\mathcal{D}_0 \leq \mathcal{D}$ whence $\mathcal{D}_0$ is non-circular. □

(1.5.7) (Ordinary) attribute grammars.

An attribute grammar is a relational attribute grammar $\langle N,P,\Phi,\mathbb{D}\rangle$ such that Attr = Inh ∪ Syn is the union of two disjoint sets (inherited and synthesized attributes) with Inh(X) = Attr(X) ∩ Inh and Syn(X) = Attr(X) ∩ Syn.

Such a splitting introduces on W(p) for each p ∈ P a partition : $W(p) = W_{in}(p) \cup W_{out}(p)$ is the disjoint union of the input and output attribute occurrences of p in $P<X_1\ldots X_n, X_0>$ defined as :

$W_{in}(p) = \{a(i) \mid a \in Inh(X_0) \text{ and } i = 0 \text{ or } a \in Syn(X_i) \text{ and } 1\leq i\leq n\}$

$W_{out}(p) = \{a(i) \mid a \in Syn(X_0) \text{ and } i = 0 \text{ or } a \in Inh(X_i) \text{ and } 1\leq i\leq n\}$

Each attribute v of $W_{out}(p)$ is defined once by a formula ϕ of the form

ϕ: $v = s\ (f1)$

where $s \in M(\mathcal{F}, \{v_1,\ldots,v_m\})$ for some signature $\mathcal{F}$ and $v_i \in W(p)$.

Φ_p is the conjuction of all attribute definitions ϕ of v in $W_{out}(p)$.

An attribute grammar is said in normal form if in(f1) $v_i \in W_{in}(p)$ only.

To every attribute grammar G corresponds an attribute dependency scheme S_G = <N,P,Attr,D> where D(p) is the local dependency graph defined as usual (see [CF82] or [Cou84]).

Certain known subclasses of attribute grammars can be characterized in terms of the properties of the corresponding dependency indicators.

(1.5.8) Proposition : *An attribute grammar* G *in normal form is strongly non-circular iff there exists a closed and non-circular dependency indicator* $\mathcal{D}$ *such that* $\mathcal{D}(X)$ *is singleton for all* X *in* N.

Proof : "Only if". One takes $\mathcal{D}(X) = \{\{(b,a)/b \in \gamma(a,X)$, a is a synthesized attribute of X$\}\}$ where γ is a closed and non-circular argument selector for G (see [CF82, p.175]).

"If". Let $\mathcal{D}$ be given.

The system of inequations (such that $r_X \subseteq \underline{Attr}(X) \times \underline{Attr}(X)$)

(f2) $r_X \supseteq \cup\{D_0^+(p)\,[r_{X_1},\ldots,r_{X_n}] \;/\; p \in P_{<X_1..X_n,X>}\}$

has the solution $(r_X^1)_{X\in N}$ where $\mathcal{D}(X) = \{r_X^1\}$ and a least solution $(r_X^0)_{X\in N}$ so that $r_X^0 \subseteq r_X^1$ for all X.

(Remark that one does not have necessarly $\cup \mathcal{D}_0(X) = r_X^0$, but only $r \subseteq r_X^0$ for all r in $\mathcal{D}_0(X)$.)

Since G is in normal form, $r_X^0 \subseteq \underline{Inh}(X) \times \underline{Syn}(X)$ (since $D_0^+(p)[r_{X_1},\ldots,r_{X_n}] \subseteq \underline{Inh}(X \times \underline{Syn}(X)$ whenever $r_{X_i} \subseteq \underline{Inh}(X_i) \times \underline{Syn}(X_i)$).

The mapping $\gamma(a,X) = \{b/(b,a) \in r_X^0\}$ is an argument selector which is closed since r_X^0 satisfies (f2).

Since $\mathcal{D}$ is non-circular and $r_X^0 \subseteq r_X^1$, this argument selector is non-circular. □

An other example is the class of attribute grammars in normal form such that the dependency indicator $\mathscr{D}_1$ such that $\mathscr{D}_1(X) = \{\underline{Inh}(X) \times \underline{Syn}(X)\}$ (which is necessarly closed by the normal form hypothesis) is non-circular.

It coincides with the class of one-visit attribute grammars [EF81] also called one-sweep in [EF82a, b, Eng84] :

(1.5.9) Proposition : *An attribute grammar* **G** *in normal form is one-sweep (equivalently one-visit) iff the dependency indicator* $\mathscr{D}_1$ *is non-circular.*

Proof : One characterization of the class of one-sweep attribute grammars is the following one :

For every p in P, the brother graph B(p) has no cycle.

The brother graph B(p) of p of profile $\langle X_1 \ldots X_n, X_0 \rangle$ is the graph with set of nodes $\{1,2,\ldots,n\}$ and which has an arc $i \longrightarrow j$ iff there exists $(a(i), b(j))$ in D(p). Since G is in normal form this condition implies $a \in \underline{Syn}(X_i)$, $b \in \underline{Inh}(X_j)$. Let $r_X = \underline{Inh}(X) \times \underline{Syn}(X)$ (i.e. $\mathscr{D}_1(X) = \{r_X\}$). We show that $D(p)[r_{X_1},\ldots,r_{X_n}]$ has no cycle if and only if B(p) has none either.

If B(p) has a cycle $i_1 \longrightarrow i_2 \longrightarrow \ldots \longrightarrow i_k \longrightarrow i_1$ there exist $a_1,\ldots,a_k$ in $\underline{Syn}(X_{i_1}),\ldots,\underline{Syn}(X_{i_k})$ and $b_1,\ldots b_k$ in $\underline{Inh}(X_{i_1}),\ldots,\underline{Inh}(X_{i_k})$ such that $(a_j(i_j), b_{j+1}(i_{j+1})) \in D(p)$ for all $j = 1,\ldots,k$ (with $i_{k+1} = i_1$, $a_{k+1} = a_1$, $b_{k+1} = b_1$).

Since $(b_j, a_j) \in r_{X_{i_j}}$ for all $j = 1,\ldots,k$, one gets a cycle in $D(p)[r_{X_1},\ldots r_{X_n}]$ of the form :

$$(*)\quad a_1(i_1) \longrightarrow b_2(i_2) \longrightarrow a_2(i_2) \longrightarrow b_3(i_3) \longrightarrow \ldots\ldots$$

Conversely, let

$$(**)\quad c_1(i_1) \longrightarrow c_2(i_2) \longrightarrow c_3(i_3) \longrightarrow \ldots\ldots \longrightarrow c_k(i_k) \longrightarrow c_1(i_1)$$

be a cycle in $D(p)[r_{X_1},\ldots,r_{X_n}]$. Observe that no i_j can be 0 : otherwise, if c_j is inherited then $c_{j-1}(i_{j-1}) \longrightarrow c_j(0)$ and this impossible since an inherited attribute is defined from the context of a node; if c_j is synthesized then

$c_j(0) \longrightarrow c_{j+1}(i_{j+1})$ violates the normal form hypothesis.

Hence the cycle (**) is of the form (*) and one gets a cycle in B(p). □

(1.5.10) Remarks :

1) For a non-circular attribute grammar G in normal form, for any closed non-circular dependency indicator $\mathcal{D}$ one has :

$$\mathcal{D}_0 \leq \mathcal{D}' \leq \mathcal{D}_1$$

where $\mathcal{D}'$ is "the restriction of $\mathcal{D}$ to $\underline{Inh}(X) \times \underline{Syn}(X)$" i.e. more formally :

$$\mathcal{D}'(X) = \{r \cap \underline{Inh}(X) \times \underline{Syn}(X) / r \in \mathcal{D}(X)\}$$

2) An attribute grammar in normal form is ℓ-ordered (Engelfriet, File [EF82b]) iff there exists a closed and non-circular dependency indicator $\mathcal{D}$ such that $\mathcal{D}(X)$ is a singleton $\{r_X\}$ and r_X is a linear (strict) order(*) on $\underline{Attr}(X)$, for all X in N.

3) All the results of this section hold if the attribute grammars are not in normal form, i.e. with local dependency graphs using relations on $W(p) \times W_{\underline{out}}(p)$.

We conclude this section with a definition extending to attribute dependency schemes the concept of L-attribute grammar.

(1.5.11) Definition

An Attribute dependency scheme D *is* <u>*of type*</u> L *(or is an* L*-attribute dependency scheme) if the set of attributes is partitioned into* <u>Inh</u> ∪ <u>Syn</u> *and, for each* p *in* P *of profile* $\langle X_1 \ldots X_n, X_0 \rangle$ *if* $(a(i), b(j)) \in D(p)$ *then the following three conditions hold :*

(1) i=0, a ∈ <u>Inh</u> *or* i ≠ 0, a ∈ <u>Syn</u>
(2) j=0, b ∈ <u>Syn</u> *or* j ≠ 0, b ∈ <u>Inh</u>
(3) if i ≠ 0 *and* j ≠ 0 *then* i < j

Hence an attribute grammar is of type L ([EF81]) iff its associated attribute dependency scheme is of type L.

(*) Thus it is a total order.

(2) Proof methods for relational attribute grammars.

We introduce specifications for attribute grammars, and the correctness of an attribute grammar w.r.t. a specification which is akin to partial correctness of programs.

We introduce a proof method allowing to establish the correctness of an attribute grammar w.r.t. a specification, which is sound. It is shown to be complete in an abstract sense i.e. if one uses the "maximum" logical language where every relation on the domain of interpretation can be represented by a formula. This completeness result is analoguous to the theorem of De Bakker and Meertens [DM75] stating the completeness of the inductive assertion method for flowcharts. But the incompleteness result of Wand [Wan 78] extends to our proof method, when one restricts formulas to first-order logic.

2.1. Specifications

(2.1.1) Definition

Given a relational attribute grammar $G = < N,P,\Phi, D >$ *, a specification for* G *consists in a family* $\theta = \{\theta^X\}_{X \in N}$ *of formulas belonging to some logical calculus* $\mathcal{L}'$ *including* $\mathcal{L}$.

Each formula has its free variables in the set $\underline{Attr}(X)$. *We shall also write* $\theta^X(a_1,\ldots,a_m)$ *to recall this, where* $\{a_1,\ldots,a_m\} = \underline{Attr}(X)$.

We say that G *is correct* w.r.t. *the specification* θ *or satisfies* θ, *or that* θ *is valid for* G *if, for all tree* t *in* M(P), *for all valid* t*-assignment* ν

$$\mathbb{D} \models \theta^X(\nu(a_1(0)),\ldots,\nu(a_m(0)))$$

where $X = \sigma(t)$.

In a practical situation, θ is given and G has to be found to satisfy θ. But for the purpose of a theoretical investigation we shall rather start from G and investigate the set of specifications which G satisfy. For this reason we shall use the upside-down terminology

"θ is valid for G".

The following proposition says that the validity of θ^X does not only hold at the root of a tree, but at all nodes, if θ is valid.

(2.1.2) Proposition : *If* θ *is valid for* G *then, for all tree* t *in* M(P), *for all valid* t*-assignment* ν

$$\mathbb{D} \models \theta^X(\nu(a_1(u)),\ldots,\nu(a_m(u)))$$

for all u *in* Node(t) *where* $X = \sigma_t(u)$ *and* $\{a_1,\ldots,a_m\} = \underline{Attr}(X)$.

Proof : Let ν, u be given as in the statement. Let $t' = t/u$; then the assignment $\nu' : W(t') \to D$ such that $\nu'(a(u')) = \nu(a(uu'))$ for all $a(u')$ in $W(t')$ satisfies $\Phi_{t'}$.
Whence the result by definition (2.1.1). □

(2.1.3) Remark :

Saying that G satisfies θ corresponds to the partial $\forall$-correctness of a program w.r.t. a specification, as defined by Gallier [Gal 81].

Other notions, in particular of total correctness are of obvious interest. In particular, one could define the total $\exists$-correctness of G w.r.t. θ by requiring the existence of at least one tree t in $M(P)_X$, at least one valid t-assignment ν such that $\nu(b_i(0)) = d_i$ for all ℓ-tuple $(d_1,\ldots,d_\ell)$ satisfying some "input-condition" $\phi^X(d_1,\ldots,d_\ell)$, where $\{b_1,\ldots,b_\ell\}$ is a subset of $\underline{Attr}(X)$. See [Der 84] for the investigation of such a concept.

(2.1.4) Definition :

Let θ and θ' be two specifications for a same relational attribute grammar G. One says that θ is weaker than θ' (θ' is stronger than θ), denoted by $\theta' \Rightarrow \theta$

if for all X,

$$\mathbb{D} \models (\theta'^X \Rightarrow \theta^X)$$

Note that θ and θ' can be written in different logical languages.
If θ is valid for G and $\theta \Longrightarrow \theta'$ then θ' is valid for G.

The specification TRUE (every formula of which is identical to true) is always valid for G in a trivial way.

It is the weakest valid specification for G.

There exists a strongest valid specification for G, belonging the language $\mathcal{L}_{\mathbb{D}}$ of all relations on D. Let us write it θ_G and define it as follows (with Attr(X)= $(a_1,\ldots,a_m)$) :

$(\mathbb{D},d_1,\ldots,d_m) \models \theta_G^X(a_1,\ldots,a_m)$ if and only if there exists t in $M(P)_X$, a valid t-assignment ν such that $\nu(a_i(0)) = d_i$ for all i=1,...,m.

In a more intuitive way :

θ_G^X is defined as $U\{R_{t,\mathbb{D}} \;/\; t \in M(P)_X\}$.

It should be noted that θ_G^X is not necessarly expressible in first-order logic (see (2.5) below).

From the definition, it is clear that a specification θ is valid for G iff it is weaker than θ_G (i.e. $\theta_G \Rightarrow \theta$).

(2.2) How to establish the correctness of an attribute grammar with respect to a specification.

(2.2.1) Definition .

Let G *be a relational attribute grammar* $\langle N,P,\Phi,\mathbb{D}\rangle$ *and* θ *be a specification for* G.

One says that θ *is inductive if, for all* p *in* P :

$$\mathbb{D} \models \Phi_p \text{ and } \theta_1^{X_1} \text{ and } \theta_2^{X_2} \ldots \text{and } \theta_n^{X_n} \Rightarrow \theta_0^{X_0}$$

where $(X_1 \ldots X_n, X_o)$ *is the profile of* p *and for every formula* $\mathcal{L}(\underline{Attr})$, $_i$ *denotes the formula* [a(i)/a ; a $\in$ Attr] *the free variables of which are in* {a(i)/a $\in$ Attr, i $\in \mathbb{N}$}.

(2.2.2) Proposition : *If* θ *is inductive then* θ *is valid for* G.

Proof : One proves by structural induction the validity of the proposition

$$\forall X \in N,\ \forall t \in M(P)_X,\ \mathbb{D} \models \Phi_t \Rightarrow \theta^X . \quad \square$$

(2.2.3) Remark :

A specification may be valid without being inductive as shown by the following example.

N = {X,Y}

P = {p,q} with p : Y → X and q : → Y

Attr(X) = {a}

Attr(Y) = {b}

Φ_p(a(0),b(1)) <=> a(0) = b(1)

Φ_q(b(0)) <=> b(0) = 0

$\mathbb{D} = \mathbb{N}$

Let θ be the specification such that

θ^X(a) <=> a = 0

θ^Y(b) <=> true

It is clearly valid but not inductive since the following does not hold in N

$\forall$a(0),b(1) [a(0) = b(1) and true => a(0) = 0]. $\square$

(2.2.4) Proposition : *The strongest valid specification for* G, *namely* θ_G, *is inductiv*

Proof : Let $p \in P_{<X_1 \ldots X_n, X_0>}$.

Let $\theta = \theta_G$. Let ν be any assignment $W(p) \to D$. We have to verify that

$$(\mathbb{D}, \nu) \models \Phi_p \text{ and } \theta_1^{X_1} \text{ and } \ldots \theta_n^{X_n} \Rightarrow \theta_0^{X_0}$$

Let us assume that the left-part of the implication is true. By definition of $\theta(= \theta_G)$ there is a tree t_i for each i=1,...,n, and a valid ν_i-assignment such that $\nu_i(a(0)) = \nu(a(i))$ for all a in Attr(X_i). Let $t = p(t_1,\ldots,t_n)$ and ν' be the t-assignment such that $\nu'(a(0)) = \nu(a(0))$ for a in Attr(X_0) and $\nu'(a(iu)) = \nu_i(a(u))$ for all a(u) in $W(t_i)$. Since ν satisfies Φ_p, ν' satisfies Φ_t hence ν' (and ν) satisty $\theta_0^{X_0}$ by the definition of θ_G. □

(2.2.5) Theorem :

Let G *be a relational attribute grammar. A specification* θ *for* G *is valid iff it is weaker than some inductive specification* θ'.

Proof and remarks : The "if" part follows from Proposition (2.2.2) It corresponds to the soundness of the proof method consisting in the following steps :

1) defining a specification θ'

2) proving that θ' is inductive

3) proving that $\theta' \Rightarrow \theta$.

The "only if" part follows form Proposition (2.2.4). It corresponds to a completeness theorem w.r.t. the language of all relations on $\mathbb{D}$. One doesnot have the completeness if one restrict assertions to some logical language like first-order logic. See (2.5) below. □

Examples of use of this proof method will be given in (3.4) and (4.2.5) below.

(2.3) Annotations

The practical usability of the above mentionned proof method suffers to its theorical simplicity : the inductive specification θ' to be found to prove the validity of some given specification θ will need complex formulas θ'^{X} since there is only one for each X in N.

In what follows, we want to describe at a syntactical and abstract level a method to break such complex formulas into Boolean combinations of simpler ones.

Roughly speaking, we shall write a formula θ^{X} as $\mathcal{M}A \Rightarrow \mathcal{M}B$ where A and B are finite sets of formulas. The formulas in A will be considered as inherited attributes and those in B as synthesized ones within a certain attribute dependency scheme. The truth of $\mathcal{M}A \Rightarrow \mathcal{M}B$ corresponds to the fact that synthesized attributes depend on inherited ones (via the subtree issued from the node). And the non-circularity of the attribute dependency scheme will insure the non-circularity of the proof of $\mathcal{M}A \Rightarrow \mathcal{M}B$, hence its truth.

This way of doing will be well suited to (usual) attribute grammars where, at each node of the tree some attribute occurrences w_0 are defined (in a unique way) in terms of neighbour attribute occurences $w_1,\ldots,w_k$. In this case, $\mathcal{M}B$ will contain some information concerning w_0, $\mathcal{M}A$ will contain some information concerning the values of the attribute occurrences at this node , upon which w_0 depends, via the subtree or the context.

In other words, the design of the formulas θ^{X} can be made in connection with the structure of the dependencies between attribute occurrences. Certain restrictions defining subclasses of attribute grammars (strongly non-circular, ℓ-ordered etc...) certainly yield specific types of formulas θ^{X}.

Such an approach has been taken by Katayama and Hoshino [KH81] for a class of attribute grammars closely connected with the class of benign attribute grammars of Mayoh [May 81].

But a lot remains to be explored concerning the other existing subclasses.

(2.3.1) Definition

Let $G = \langle N,P,\Phi,\mathbb{D}\rangle$ be a relational attribute grammar.

An annotation of G is a mapping Δ assigning to every X in N a finite set $\Delta(X)$ of formulas of $\mathscr{L}$ (Attr(X)) where $\mathscr{L}'$ is some logical language containing $\mathscr{L}$.

The set $\Delta(X)$ is partitionned into two sets $I\Delta(X)$ (the set of inherited assertions of X) and $S\Delta(X)$ (the set of synthesized assertions of X).

The specification θ_Δ associated with Δ is the one such that θ_Δ^X is the formula :

$$\mathbb{M}\, I\Delta(X) \Rightarrow \mathbb{M}\, S\Delta(X)$$

We say that Δ is valid (resp. inductive) if θ_Δ has the same property.

It is clear that every specification can be seen as the annotation Δ_θ such that $S\Delta(X) = \{\theta^X\}$ and $I\Delta(X) = \{\text{true}\}$ and that θ_{Δ_θ} coincides with θ.

We shall now give sufficient conditions insuring the validity of an annotation.

(2.3.2) Soundness of an annotation.

Let G and Δ be as above.

Let D be a mapping associating with p in $P_{<X_1 \ldots X_n, X_0>}$ a graph D(p) with set of nodes :

$$V(p) = \{\phi(i) \;/\; 0 \le i \le n \;,\; \phi \in \Delta(X_i)\}$$

(i.e. V(p) is defined w.r.t. Δ as W(p) is w.r.t. Attr) such that the target of every arc belongs to :

$$V_{concl}(p) = \{\phi(i) \in V(p) \;/\; \phi \in S\Delta(X_0) \text{ and } i = 0$$

$$\text{or } \phi \in I\Delta(X_i) \text{ and } 1 \le i \le n_p\}$$

Letting $\underline{\Delta}$ denote $U\{\Delta(X)/X \in N\}$ this means that $S = \langle N,P, \underline{\Delta}, D\rangle$ is an attribute dependency scheme.

We shall be interested in non-circular attribute dependency schemes.

We say that Δ is D-sound if for all p in P (of profile $\langle X_1 \ldots X_n, X_0\rangle$), for all $\phi(i)$ in $\underline{V}_{\underline{Concl}}(p)$, the following formula (*) is valid in $\mathbb{D}$:

(*) Φ_p and $M\{\psi_j/\psi(j) \in V(p), (\psi(j),\phi(i)) \in D(p)\} \Rightarrow \phi_i$

(where the mapping $\psi(j) \to \psi_j$ is as in def. (2.2.1))

(2.3.3) Remark

We shall also assume the following condition, (saying that D is in normal form) :

Every arc of D(p) has its source in $V(p) - V_{concl}(p)$.

This condition is not a restriction since every non-circular attribute dependency scheme D can be transformed into one D' which is non-circular and in normal form and such that D'(p) is a subgraph of the transitive closure of D(p). From this fact it follows that if Δ is D-sound then it is D'-sound, since the implications (*) extend by transitivity to the corresponding ones for D'.

Finally, if $\mathscr{D}$ is a dependency indicator for D which is closed and non-circular then so is $\mathscr{D}'$ such that $\mathscr{D}'(X) = \{r \cap \underline{Inh}(X) \times \underline{Syn}(X)/r \in \mathscr{D}(X)\}$ for D'.

(2.4) The use of annotations for correctness proofs.

We shall prove that if Δ is D-sound and S is non-circular then θ_Δ is valid. A slight difficulty comes from the fact that θ_Δ is not necessarly inductive.

To do the proof we shall associate with Δ another specification θ which is inductive and stronger than θ_Δ. This is the purpose of the following definition.

(2.4.1) Dependency indicators and specifications.

Let G, Δ be as above.

Let $\mathscr{D}$ be a dependency indicator.

Let $\theta_{\mathscr{D}}$ be the specification associated with $\mathscr{D}$ as follows :

$\theta^X_{\mathscr{D}}$ is $W\{\theta^X_r \;/\; r \in \mathscr{D}(X)\}$

θ^X_r is $M\{\theta^X_{r,\phi} \;/\; \phi \in S\Delta(X)\}$

$\theta^X_{r,\phi}$ is $M\{\psi \in \Delta(X) \;/\; (\psi,\phi) \in r\} \Rightarrow \phi$

(2.4.2) Proposition : *1) If $\mathscr{D} \leq \mathscr{D}'$ then $\theta_{\mathscr{D}} \Rightarrow \theta_{\mathscr{D}'}$.*

2) If $\mathscr{D}$ is closed and non-circular then $\theta_{\mathscr{D}}$ is inductive (hence valid).

Proof : (1) If $r \subseteq r'$ then $\theta^X_{r,\phi} \Rightarrow \theta^X_{r',\phi}$ hence $\theta^X_r \Rightarrow \theta^X_{r'}$.

Since $\mathscr{D} \leq \mathscr{D}'$, if $\mathscr{D}(X) = \{r_1,\ldots,r_k\}$ there exists a subset $A = \{r'_1,\ldots,r'_{k'}\}$ of $\mathscr{D}'(X)$ such that for all i=1,...,k there exists j in 1,...,k' such that $r_i \subseteq r'_j$. Hence

$$\theta^X_{\mathscr{D}} \Rightarrow W\{\theta^X_{r'_i} \;/\; 1 \leq i \leq k\}$$

$$\Rightarrow \theta^X_{\mathscr{D}'}.$$

Hence we have shown that $\theta_{\mathscr{D}} \Rightarrow \theta_{\mathscr{D}'}$.

(2) We have to show that for p in $P_{<X_1\ldots X_n,X_o>}$ the following formula hold (i.e. is valid in $\mathbb{D}$; this terminology will be used in the sequel of the proof) :

$$\Phi_p \text{ and } \theta^{X_1}_1 \text{ and } \ldots \theta^{X_n}_n \Rightarrow \theta^{X_o}_o \text{ where } \theta = \theta_{\mathscr{D}}$$

It suffices to show that for all r_1 in $\mathscr{D}(X_1),\ldots,r_n$ in $\mathscr{D}(X_n)$ there exists r_o in $\mathscr{D}(X_o)$ such that the formula

$$\Phi_p \text{ and } \theta^{X_1}_{r_1,1} \text{ and } \ldots \text{ and } \theta^{X_n}_{r_n,n} \Rightarrow \theta^{X_o}_{r_o,0}$$

holds. We show this by taking some r_o such that $D^+_o(p)[r_1,\ldots,r_n] \leq r_o$ (since $\mathscr{D}$ is closed). We need only show that for all ϕ in $S\Delta(X_o)$, the formula

$$\Phi_p \text{ and } \theta^{X_1}_{r_1,1} \ldots \theta^{X_n}_{r_n,n} \text{ and } \mathbb{M}\{\psi_o / (\psi,\phi) \in r_o\} \Rightarrow \phi_o$$

holds. Let us abbreviate into Ψ the left-hand side of this implication.

Let $H = D(p)[r_1,\ldots,r_n]$. Let K be the set of vertices of H consisting of $\phi(0)$ and all its predecessors in H (i.e. the vertices of H from which there is a path to $\phi(0)$).

We want to show that

(*) $\Psi \Rightarrow \eta(k)$

for all $\eta(k)$ in K.

Since H has no cycle , it suffices to show that (*) holds provided it holds for the immediate predecessors of $\eta(k)$ in H.

We do this by considering several cases :

Case 1 : $\eta(k) \in V_{concl}(p)$

The result follows the hypothesis that Δ is D-sound (see definition (2.3.2)).

Case 2 : $\eta(k) \notin V_{concl}(p)$, $k = 0$. $(\eta(0) \in I\Delta(X_o))$

In this case $(\eta(0),\phi(0)) \in H^+$ hence $(\eta,\phi) \in r_o$; then $\Psi \Rightarrow \eta(0)$ holds in a trivial way since Ψ is of the form Ψ' and ψ_o with $\psi_o = \eta(0)$.

Case 3 : $\eta(k) \notin V_{concl}(p)$, $1 \leq k \leq n$.

Hence $\eta \in S\Delta(X_k)$.

The immediate predecessors of $\eta(k)$ are the $\psi(k)$'s such that $(\psi,\eta) \in r_k$.

The result follows from the fact that Ψ is of the form Ψ' and $(\mathbb{M}\{\psi_k \in \Delta(X_k) / (\psi,\eta) \in r_k\} \Rightarrow \eta_k)$. □

(2.4.3) Theorem :

Let G be a relational attribute grammar. If an annotation Δ is D-sound for some non-circular attribute dependency scheme D, it is valid. □

Proof : Let $\mathscr{D}$ be any closed and non-circular dependency indicator for D (there exis some since D is non-circular, in particular the canonical one $\mathscr{D}_0$, and possibly other simpler ones). By Prop. (2.4.2) - 2) the specification $\theta_{\mathscr{D}_0}$ is inductive hence valid.

The specification θ_Δ is nothing else than $\theta_{\mathscr{D}_1}$ associated with the dependency indicator $\mathscr{D}_1$ of Prop.(1.5.9). Let $\mathscr{D}'$ be associated with $\mathscr{D}$ by (2.3.3). Then by Prop.(2.4.2)-1) and since $\mathscr{D}' \leq \mathscr{D}_1$ (see (1.5.10)), $\theta_{\mathscr{D}'} \Rightarrow \theta_\Delta$ hence θ_Δ is valid (since $\theta_{\mathscr{D}'}$ is inductive and valid). □

(2.4.4) Remark :

The proof of theorem (2.4.3) shows how an inductive specification $\theta_{\mathscr{D}}$ can be used instead of a D-sound annotation (where D is non-circular), to prove some valid specification.

The specification $\theta_{\mathscr{D}_0}$ is quite complex. Relatively simple specifications $\theta_{\mathscr{D}}$ are obtained if $\mathscr{D}(X)$ is singleton for all X. This is the case if D is strongly non-circular (see Prop.(1.5.8), or if D is one-sweep (1.5.9) and in this last case the inductive specification $\theta_{\mathscr{D}_1}$ coincides with θ_Δ.

Note that since $\mathscr{D}$ is not a singleton in general, the inductive assertions of $\theta_{\mathscr{D}}$ can have a size which is exponential in the size of S, counted as $\Sigma\{\underline{Card}(\Delta(X))/X \in N\}$. Hence the complete inductive proof on G is of "exponential complexity" whenever the verification of the soundness of Δ is of "linear complexity" in th

size of <N,P, $\underline{\Delta}$,D>. This shows precisely how the introduction of annotations decreases the complexity of the correctness proof.

(2.4.5) Example.

We define a relational attribute grammar, an annotation Δ which is D-sound but such that θ_Δ is not inductive. In fact we only define D and display it as a set of dependency graphs in a classical way (see for instance [CF82]).

$N = \{X,Y\}$, $P = \{p,q,r\}$

$I\Delta(X) = \{\alpha\}$, $S\Delta(X) = \{\beta\}$, $I\Delta(Y) = \{\gamma,\mu\}$, $S\Delta(Y) = \{\eta,\delta\}$

D(p), D(q), D(r) are shown on Fig.1, 2, 3 respectively.

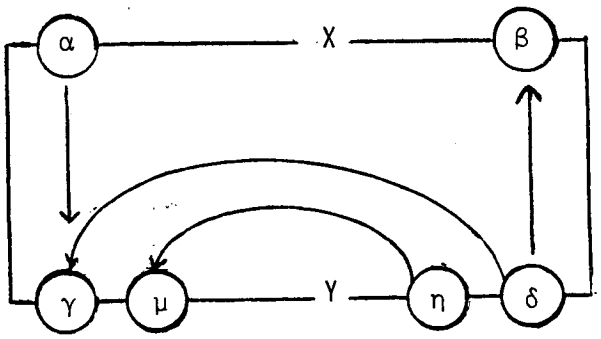


Fig.1 (p)

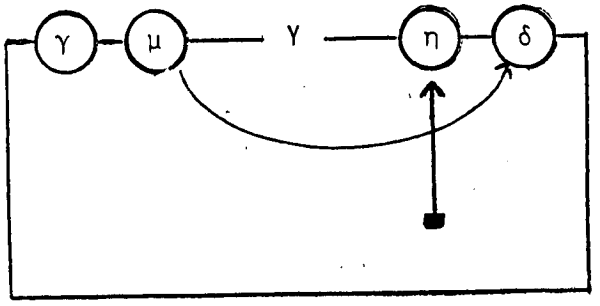


Fig.2 (q)

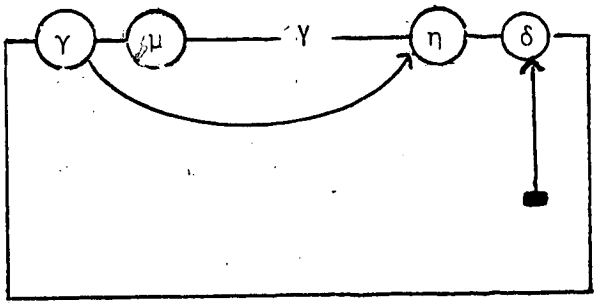


Fig.3 (r)

The corresponding attribute dependency scheme is non-circular but not strongly non-circular.

We assume that Δ is D-sound i.e. that the following formulas are valid :

(1) Φ_p and δ => β

(2) Φ_p and α and δ => γ

(3) Φ_p and η => μ

(4) Φ_q => η

(5) Φ_q and μ => δ

(6) Φ_r => δ

(7) Φ_r and γ => η

The specification $\theta = \theta_\Delta$ is valid but not inductive . Its formulas are :

θ^X : α => β

θ^Y : γ and μ => η and δ

Saying that θ_Δ is inductive would mean that :

(8) Φ_p and (γ and μ => η and δ) => (α => β)

(9) Φ_q => (γ and μ => η and δ)

(10) Φ_r => (γ and μ => η and δ)

Whereas (9) and (10) easily follow from (4), (5), (6) and (7), (8) does not follow from (1)-(7).

It suffices to take Φ_p, α, η, μ equal to the constant Boolean value *true* and β, γ, δ equal to *false* to verify this fact.

Here is the specification $\theta' = \theta_{\mathcal{D}_0}$

θ'^X : β

θ'^Y : (η *and* (μ => δ)) *or* (δ *and* (γ => η))

which is both inductive and stronger than θ. □

(2.5) Wand's incompleteness result

We show an example of a relational attribute grammar and a valid specification for which no inductive specification written in first-order logic can be found to prove it. It follows from Prop. (2.2.4) that no sound annotation in first-order logic can be found either.

Our example is a straightforward translation of an example of Wand [Wan78] showing the incompleteness of Hoare's logic when invariant assertions are written in first-order logic.

Let p, q, r be unary predicates, f be a unary function symbol, x, y be variables.

Let D be the first-order structure such that :

its domain D is $\{a_n \,/\, n \geq 0\} \cup \{b_n \,/\, n \geq 0\}$

$f(a_0) = a_0$, $f(b_0) = b_0$,

$f(a_n) = a_{n-1}$, $f(b_n) = b_{n-1}$ for $n \geq 1$,

$p(x)$ = *true* iff $x = a_0$

$q(x)$ = *true* iff $x = b_0$

$r(x)$ = *true* iff $x = a_n$ for some n of the form $k(k+1)/2$

It is shown in [Wan78, theorem 2] that there is no first-order formula ϕ with one free variable x such that, for d in D, $(\mathbb{D},d) \models \phi$ iff $d \in \{a_n / n \geq 0\}$.

Let G be the following relational attribute grammar :

$N = \{A,X\}$

$P_{<X,A>} = \{a\}$, $P_{<X,X>} = \{b\}$, $P_{<\varepsilon,X>} = \{c\}$

$\underline{Attr}(A) = \underline{Attr}(X) = \{x,y\}$

Φ_a is the formula : $r(x(0))$ and $x(0) = x(1)$ and $y(0) = y(1)$

Φ_b is the formula : not $(p(x(0)))$ and (not $(q(x(0))))$

and $x(1) = f(x(0))$ and $y(0) = y(1)$

Φ_c is the formula : $(p(x(0))$ or $q(x(0)))$ and $x(0) = y(0)$

$\mathbb{D}$ is the above defined interpretation.

It is easy to see that the strongest specification θ_G is the following :

$\theta_G^X(x,y)$ iff $(x \in \{a_n / n \geq 0\}$ and $y = a_0)$

or $(x \in \{b_n / n \geq 0\}$ and $y = b_0)$

$\theta_G^A(x,y)$ iff $(x \in \{a_n / n = k(k+1)/2$ for some $k \geq 0\}$

and $y = a_0$.

Let θ be the valid specification such that :

$\theta^X(x,y)$ is true

$\theta^A(x,y)$ is $p(y)$.

Assume there is an inductive specification θ' written in first-order logic such that $\theta' \Rightarrow \theta$. It must satisfy the following conditions : (theorem (2.2.5) and def. (2.2.1)

(1) $\theta'^{A}(x,y) \Rightarrow p(y)$

(2) $\theta'^{X}(x,y)$ and $r(x) \Rightarrow \theta'^{A}(x,y)$

(3) $\theta'^{X}(fx,y)$ and (not $(p(x))$) and (not $(q(x))$) $\Rightarrow \theta'^{X}(x,y)$

(4) $(p(x)$ or $q(x))$ and $x = y \Rightarrow \theta'^{X}(x,y)$

Let now ϕ be the formula $\forall y[\theta'^{A}(x,y) \Rightarrow p(y)]$

Claim : $(\mathbb{D},d) \models \phi$ iff $d \in \{a_n \mid n \geq 0\}$.

We first note some facts.

From (4) :

(5) $\theta'^{X}(b_o,b_o)$ holds.

From (5) and (3) the following holds :

(6) $\theta'^{X}(b_n,b_o)$ for all $n \geq 1$.

If $\theta'^{X}(a_n,d')$ holds then $\theta'^{X}(a_m,d')$ holds for all $m \geq n$ (by (3)). Let us choose $m \geq n$ of the form $k(k+1)/2$; then $\theta'^{A}(a_m,d')$ holds by (2) hence $p(d')$ holds by (1).

This shows that $(\mathbb{D},d) \models \phi$ for all d in $\{a_n \mid n \geq 0\}$.

Let now $d = b_n$ for any n. Then $\theta'^{X}(b_n,b_o)$ holds by (5) and (6) ; since $p(b_o)$ does not, $(\mathbb{D},d) \not\models \phi$.

The claim is proved. But ϕ (hence θ') cannot be first-order formulas. Contradiction. □

(3) Application to recursive imperative procedures.

We show that the semantics of recursive imperative procedures can be formalized by means of attributes associated to the nodes of a tree called the tree of calls which defines the structure of recursive calls in some computation for some interpretation and some values of the arguments.

We shall apply this formalization to the problem of proving the partial correctness of a recursive imperative program by the inductive assertion method, by using the method introduced in Section 2.

In order to give precise statements, we need a precise class of programs.

We shall use the one introduced by Gallier [Gal81] where the provability of partial correctness has also been investigated.

(3.1) Recursive imperative procedures

(3.1.1) Definitions.

Let $\mathscr{F}$, $\mathscr{R}$ and $\mathscr{V}$ be as in (1.3). Since the set of sorts $\mathfrak{a}$ will be irrelevant we shall simply take it reduced to a single sort. The extension to multiple sorts is trivial.

The set $\mathscr{V}$ will be partitioned into $\mathscr{V}_I \cup \mathscr{V}_L \cup \mathscr{V}_0$ where $\mathscr{V}_I = \{x_1, x_2, \ldots, x, x', x'', \ldots\}$ is the set of input variables, $\mathscr{V}_L = \{y_1, y_2, \ldots, y, y', \ldots$ is the set of local variables and $\mathscr{V}_0 = \{z_1, z_2, \ldots, z, z', \ldots\}$ is the set of ouput variables.

We shall denote by $\mathscr{V}_{I,k}$ the set $\{x_1, x_2, \ldots, x_k\}$ and similarly for $\mathscr{V}_{L,k}$, $\mathscr{V}_{0,k}$.

Let $\mathscr{A}$ be a finite set of procedure symbols ; each A in $\mathscr{A}$ has a rank $\rho(A)=(n,m$ which is a pair of positive integers.

A procedure definition is an equation

$$A(x_1, \ldots, x_n \; ; \; z_1, \ldots, z_m) = S_A$$

where $(n,m) = \rho(A)$ and S_A is a flowchart over $(\mathscr{F}, \mathscr{R}, \mathscr{A})$ of rank (n,m).

We shall abbreviate it into

$$A(\bar{x};\bar{z}) = S_A$$

where $\bar{x}$ represents the list $(x_1,\ldots,x_n)$ called the list of formal input parameters and $\bar{z}$ represents $(z_1,\ldots,z_m)$, the list of formal output parameters.

A flowchart S over $(\mathscr{F},\mathscr{R},\mathscr{A})$ and of rank (n,m) consists of

(1) a finite directed graph G with a special node in called the entry node, one special node out called the exit node and such that in (resp. out) is not the target (resp. the source) of any edge and such that each node belongs to some path from in to out.

(2) a labelling function which associates an instruction over $(\mathscr{F},\mathscr{R},\mathscr{A})$ of type (n,p,m) with every edge, and satisfying a condition which will be stated later in terms of computation paths.

By an instruction of type (n,p,m) over $(\mathscr{F},\mathscr{R},\mathscr{A})$ we mean :

(1) either null, the null instruction

(2) or a simultaneous assignment of the form :

$$(v_1,\ldots,v_k) \leftarrow (t_1,\ldots,t_k)$$

where $v_1,\ldots,v_k$ are pairwise distinct variables in $\mathscr{V}_{0,m} \cup \mathscr{V}_{L,p}$ and $t_1,\ldots,t_k \in \mathscr{F}(\mathscr{V}_{I,n} \cup \mathscr{V}_{L,p}) \cup \mathscr{V}_{I,n} \cup \mathscr{V}_{L,p}$ (*)

(3) or a guard i.e. an instruction of the form :

$$R(v_1,\ldots,v_k)$$

or

$$\text{not } (R(v_1,\ldots,v_k))$$

(*) Elementary terms

for some R in $\mathscr{R}_k$, $v_1,\ldots,v_k$ in $\mathscr{V}_{I,n} \cup \mathscr{V}_{L,p} \cup \mathscr{V}_{0,m}$.

(4) or a procedure call i.e. an instruction of the form

call $A(u_1,\ldots,u_{n'} ; v_1,\ldots,v_{m'})$

where $A \in \mathscr{A}$, $\rho(A) = (n',m')$, $u_1,\ldots,u_{n'} \in \mathscr{V}_{I,n} \cup \mathscr{V}_{L,p}$, $v_1,\ldots,v_{m'} \in \mathscr{V}_{L,p} \cup \mathscr{V}_{0,m}$ and $v_i \neq v_j$ for $i \neq j$ and $v_i \neq u_j$ for all i,j.

Such a procedure call is abbreviated call $A(\bar{u};\bar{v})$; $\bar{u}(\bar{v})$ is the list of actual input (output) parameters.

A system of recursive imperative procedures Σ over $(\mathscr{F},\mathscr{R},\mathscr{A})$ is a set of procedure definitions over $(\mathscr{F},\mathscr{R},\mathscr{A})$ such that for all A in $\mathscr{A}$ one has one and only one definition with left-hand side of the form $A(\bar{x};\bar{z})$.

A recursive imperative program scheme over $(\mathscr{F},\mathscr{R})$ (we shall simply say a scheme in this section), is a pair $<\Sigma,A>$ consisting of a system of recursive imperative procedures Σ over $(\mathscr{F},\mathscr{R},\mathscr{A})$ and some A in $\mathscr{A}$ playing the role of the "main program".

An interpretation for (Σ,A) (or Σ) is any structure $I = <D,(f_I)_{f\in\mathscr{F}},(R_I)_{R\in\mathscr{R}}>$ where the f_I's and the R_I's are total functions and relations over D of appropriate arity.

A recursive imperative program P is a pair $<<\Sigma,A>, I>$ consisting of scheme and an interpretation.

Such a program defines a relation $P_I \subseteq D^n \times D^m$ where $\rho(A) = (n,m)$. The definition of P_I will be recalled informally in section (3.2) below. A formal definition can be found in Gallier [Gal81]

(3.1.2) Example.

Let Σ be reduced to the single definition $A(x_1,x_2;z) = S$ where S is the flowchart shown on Fig. 4 .

In order to help the reader to make a correspondence between Fig 4. and definition (3.1.1) we precise that the nodes of the underlying graph are in, α, β, γ, δ, ε , out. The instructions labelling the edges are indicated in boxes.

The node α is a non-deterministic choice. The node β is due to the presence of the two exclusive guards $p(x_1,y_2)$ and not $(p(x_1,y_2))$: at most one of the edges (β,δ) and (β,γ) can be taken in each case. □

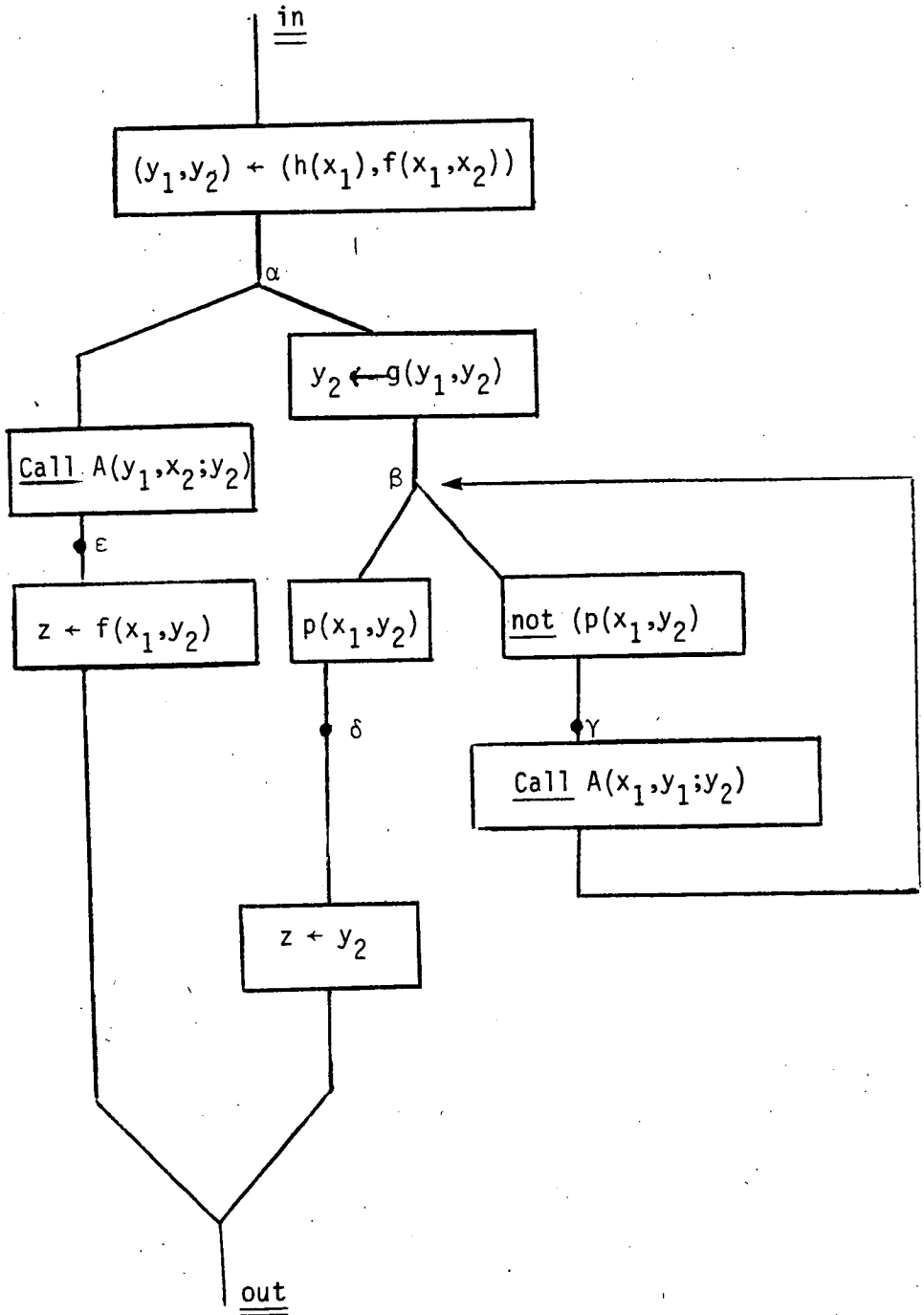


Fig.4 : A $(x_1, x_2 ; z)$

(3.1.3) Syntactic restrictions.

The above definitions reproduce Gallier's ones except that we have omitted the relational assignments (just for simplifying the exposition but all our definitions and results could be extended so as to admit them).

Here we make another more important restriction : a flow-chart is loop-free if its underlying graph has no cycle.

It is well-known that a scheme $\langle\Sigma,A\rangle$ where some right-hand sides of procedure definitions are not loop-free can be transformed into an equivalent one with loop-free right hand-sides at the cost of introducing new procedure symbols.

A special case of this transformation is the recursive definition of the while construct.

We only recall it informally on example (3.1.4).

(3.1.4) Example. (continuation of example (3.1.2)).

We claim that $\langle\Sigma,A\rangle$ is equivalent to $\langle\Sigma',A\rangle$ where Σ' consists of the two definitions :

$A(x_1,x_2;z) = S'$

$B(x_1,x_2,x_3;z) = T$

where S' and T are shown of Figures 5 and 6 respectively.

Procedure B replaces the computations made in S on the paths from β to δ. Its formal input parameters represent the values of the variables x_1, y_1, y_2 in S and the formal output parameter z represents the value of y_2, which is the only variable of S the value of which can be modified in S between β and δ.

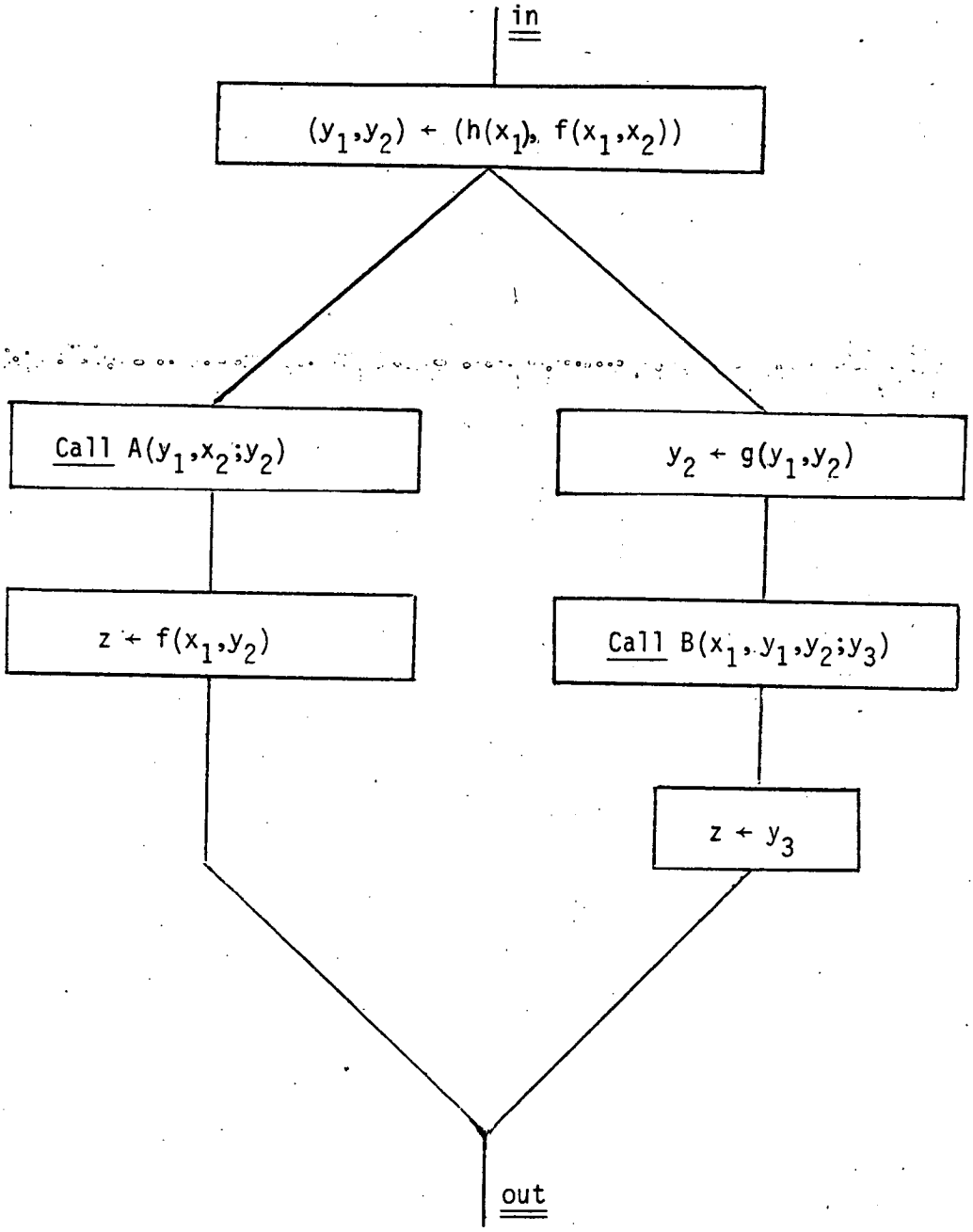

in
$(y_1,y_2) \leftarrow (h(x_1), f(x_1,x_2))$
Call $A(y_1,x_2;y_2)$
$y_2 \leftarrow g(y_1,y_2)$
$z \leftarrow f(x_1,y_2)$
Call $B(x_1,y_1,y_2;y_3)$
$z \leftarrow y_3$
out

Fig.5 : A $(x_1,x_2;z)$

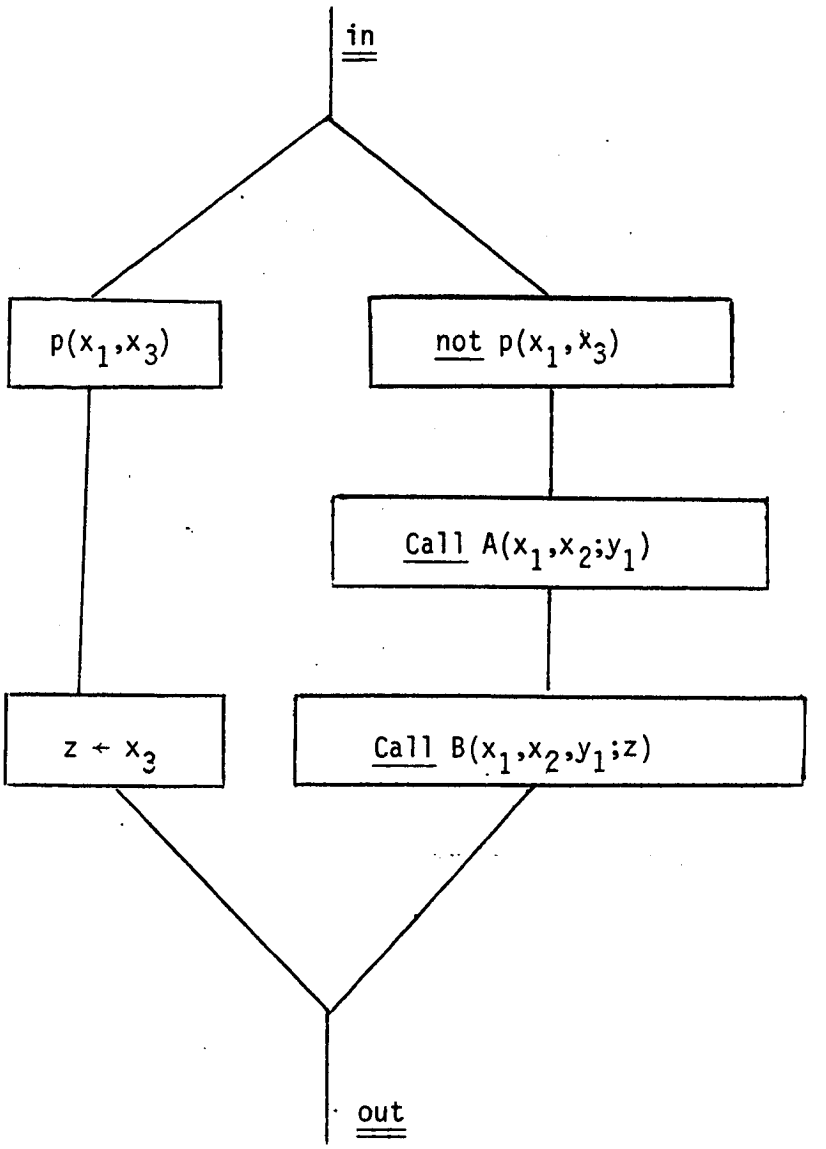


Fig.6 : $B(x_1,x_2,x_3;z)$

Let us now consider B : it uses one local variable y_1 which handles the value of y_2 in S after the call to A at edge (γ,β). Hence y_1 appears as the third actual input parameter in the call to B of B. Note also the assignment $z \leftarrow x_3$ in B necessary by the convention that output variables are distinct from input variables in procedures. □

(3.2) Operational semantics

(3.2.1) Informal definition.

We recall the definition given by Gallier [Gal81,pp 204-209].

We first consider the case of a flowchart S of rank (n,m) over $(\mathscr{F},\mathscr{R})$ i.e. without procedure calls (but with possible loops).

Let Paths(S) be the set of all finite paths in S from in to out. Such a path can be formally defined as a sequence

$$p = (\underline{in}, \theta_1, s_1, \theta_2, s_2, \ldots, \theta_k, \underline{out})$$

where the s_i's are its nodes and θ_i is the instruction labelling the edge (s_{i-1}, s_i) of p.

Let I be an interpretation (with domain D) and $\bar{d}$ be an n-tuple of input values (in D). If there exists a successful computation of S in I with input $\bar{d}$, it must follow some path in Paths(S) and yields as a result an m-tuple $\bar{d}'$.

The set of pairs $(\bar{d},\bar{d}')$ in $D^n \times D^m$ associated in this way with some path p in Paths(S) will be denoted by p_I (note that p_I is functional).

The relation S_I defined by S in I is the union of the p_I's for p in Paths(S). Since we allow nondeterministic choices, it is not functional in general. Note also that p is a flowchart, with only one path from in to out.

The formal definition of p_I from p is obvious (we hope) so we omit it.

For a flowchart S over $(\mathscr{F},\mathscr{R},\mathscr{A})$, we shall denote by Paths(S) the same set as before. Some of the θ_i's occuring will be procedure calls i.e. instructions of the form call $A(\bar{u};\bar{v})$.

We denote by $S_{I,0}$ the union of all p_I's for p in $\text{Paths}_0(S)$ defined as the set of all paths in Paths(S) which do not contain any procedure call. This means that $S_{I,0}$ is the relation defined by S if one assumes that no procedure call terminates.

Of course our aim is to define S_I, the relation defined by S in I in the context of a system $\Sigma = \langle A(\bar{x};\bar{z}) = S_A \ ; \ A \in \mathcal{A} \rangle$.

In Gallier [Gal 81] , Σ is considered as a graph grammar with productions $A \to S_A$, where applying a production at some edge labeled by call $A(\bar{u};\bar{v})$ consits in substituting the "body" S_A of A for its "call", in a way which formalizes Algol's "copy rule". The notation $S \overset{*}{\to} T$ will be used if T is obtained from S after finitely many such replacements.

Then S_I is defined by :

$$S_I = U \{T_{I,0} \ / \ S \overset{*}{\to} T\}.$$

Let us precise that the substitution of S_A for call $A(\bar{u};\bar{v})$ involves :

- a substitution of actual parameters $(\bar{u},\bar{v})$ for the formal ones in S_A

- a renaming of the local variable of S_A, $\bar{y}$ with a "fresh" copy $\bar{y}'$ of $\bar{y}$.

More formally the result T of this substitution can be written :

$$T = S[S'_A \ / \ e]$$

where e is an edge labeled by call $A(\bar{u};\bar{v})$ and

$$S'_A = (S_A[\bar{y}'/\bar{y}])[\bar{u}/\bar{x} \ , \ \bar{v}/\bar{z}]$$

where $\bar{y}$ is the p-tuple of local variables of S_A and $\bar{y}'$ is a p-tuple of distinct variables which appear neither in S nor in $\bar{x}$ nor in $\bar{z}$.

By $S[S'_A/e]$ we mean the substitution of S'_A for e in S i.e. if e has source s and target s', the deletion of e and the identification of s(s') with the node in (out) of S'_A.

By $S_A[\bar{y}'/\bar{y}]$, we denote the simultaneous substitution of y'_i for each occurrence of y_i, where $\bar{y} = (y_1,\ldots,y_p)$ and $\bar{y}' = (y'_1,\ldots y'_p)$.

Here is an example showing the need for the renaming of $\bar{y}$ into $\bar{y}'$ in S_A.

(3.2.2) Example.

Let us consider the recursive definition $A(x;z) = S_A$ where S_A is shown on figure 7.

Let T such that $S_A \rightarrow T$ be shown on figure 8.

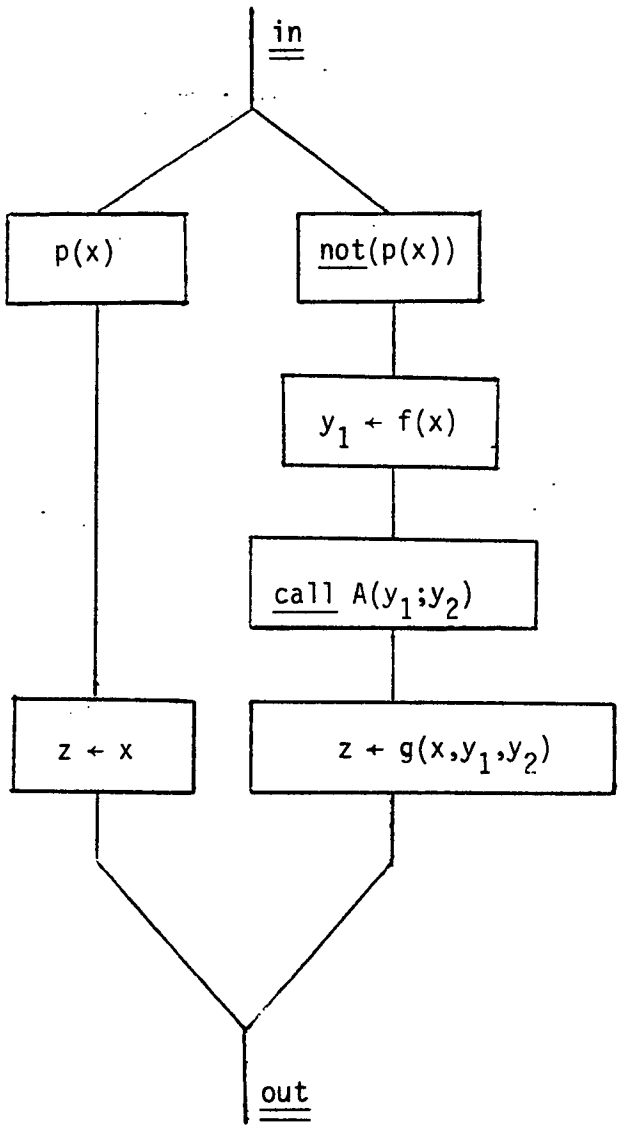


Fig. 7 A(x;z)

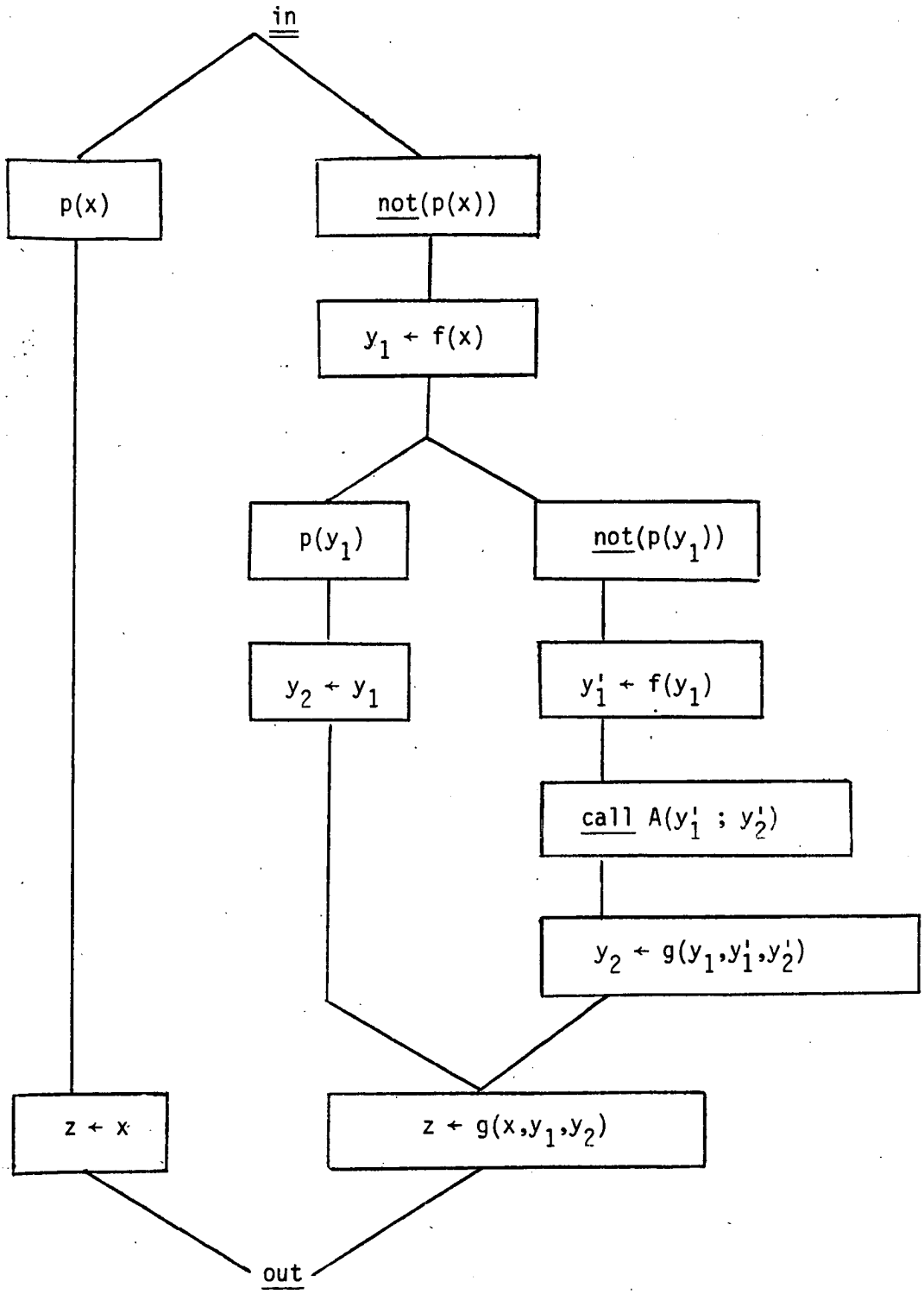

in
p(x)
not(p(x))
$y_1 \leftarrow f(x)$
$p(y_1)$
not($p(y_1)$)
$y_2 \leftarrow y_1$
$y_1' \leftarrow f(y_1)$
call $A(y_1' ; y_2')$
$y_2 \leftarrow g(y_1, y_1', y_2')$
$z \leftarrow x$
$z \leftarrow g(x, y_1, y_2)$
out

Fig. 8 T

Observe in T the use of (y_1', y_2') which renames (y_1, y_2). With $y_1' = y_1$ and $y_2' = y_2$ the semantics would have been different. □

Our use of attribute grammars for defining S_I will eliminate the renaming of local variables. In fact, the local variables themselves will disappear.

(3.2.3) Well-formed program schemes.

We now state the extra condition announced in (3.1) that our flowcharts will have to satisfy.

It corresponds to requiring that every variable has been assigned a value being used. This condition will be stated in terms of computation paths.

Let S be a flowchart of rank (n,m) over $(\mathscr{F}, \mathscr{R}, \mathscr{A})$.

Let $p = (\underline{\underline{in}}, \theta_1, s_1, \theta_2, \ldots, \theta_k, \underline{\underline{out}})$ be a path in S, let v be a variable and $i \in \{0,\ldots,k\}$. One says that v is defined at s_i (with $s_0 = \underline{\underline{in}}$, $s_k = \underline{\underline{out}}$) iff :

- either $v \in V_{I,n}$
- or $v \in V_L \cup V_0$ and v is in the left-hand side of some assignment θ_j with $j \leq i$ or in the actual output parameter list of some procedure call θ_j with $j \leq i$.

A variable v is needed at s_i iff :

- either $i = k$ (i.e. $s_i = \underline{\underline{out}}$) and $v \in V_{0,m}$
- or $i < k$ and θ_{i+1} is a guard where v occurs or θ_{i+1} is an assignment and v occurs in its right-hand side or θ_{i+1} is a procedure call and v occurs in its actual input parameter list.

A flowchart is well-formed if for all p in Paths(S), all node s in p, all variable which is needed at s is defined at this node.

This condition can be tested by an algorithm. This decidability result is obvious for a loop-free flowchart S since Paths(S) is finite.

In the sequel, every flowchart will be assumed well-formed.

(3.3) Operational semantics via attribute grammars.

(3.3.1) The tree of calls of a procedure evaluation.

Let us consider a loop-free system

$\Sigma = \langle A(\bar{x};\bar{z}) = S_A \ ; \ A \in \mathcal{O}\mathcal{A} \rangle$.

Remark first that $\underline{Paths}(S_A)$ is finite for all A.

Let $P = \cup \{\underline{Paths}(S_A) \ / \ A \in \mathcal{O}\mathcal{A}\}$ (one insures that $\underline{Paths}(S_A) \cap \underline{Paths}(S_B) = \emptyset$ if $A \neq B$ by taging with A the paths of S_A)

One turns P into an $\mathcal{O}\mathcal{A}$-signature by letting $\sigma(p) = A$ if $p \in \underline{Paths}(S_A)$ and $\alpha(p) = A_1 A_2 \ldots A_k$ being the list procedure names called in p (formally if $p = (\underline{in}, \theta_1, s_1, \ldots, \theta_\ell, \underline{out})$ then $\alpha(p) = \alpha_1 \alpha_2 \ldots \alpha_\ell$ with $\alpha_i = A$ if θ_i is $\underline{call}\ A(\bar{u};\bar{v})$ for some $\bar{u}$, $\bar{v}$ and $\alpha_i = \varepsilon$ (the empty word) otherwise).

We say that a tree in $M(P)_A$ is a tree of calls of A.

Such a tree defines the structure of recursive calls in some computation. The variables will be considered as attributes associated with procedures names, and appropriate semantic rules derived from Σ will represent their changes of value during the computation.

Furthermore, local variables will disappear ; they will be symbolically evaluated into terms depending on input variables.

(3.3.2) Construction of an attribute grammar.

For A in $\mathcal{O}\mathcal{A}$ of rank (n,m) we let $\underline{Inh}(A) = \{x_1, x_2, \ldots, x_n\}$ (its set of formal input parameters), $\underline{Syn}(A) = \{z_1, z_2, \ldots, z_m\}$ (its set of formal output parameters), and $\underline{Attr}(A) = \underline{Inh}(A) \cup \underline{Syn}(A)$.

Let us also assume that $\{y_1, \ldots, y_q\}$ is the set of local variables of S_A.

We now define Φ_p for each p in $\underline{Paths}(S_A)$. This formula will be a conjunction of a set of Boolean conditions and of a set of equations Γ_p following the usual restrictions concerning the definitions of attributes, so that the relational attribute grammar we shall define is very close to an ordinary attribute grammar.

Let $p = (s_0, \theta_1, s_1, \ldots, \theta_k, s_k) \in \underline{Paths}(S_A)$ with $s_0 = \underline{\underline{in}}$ and $s_k = \underline{\underline{out}}$. Let $r_1, r_2, \ldots, r_\ell$ be the list of indices i such that θ_i is a call and is of the form $A_i(\bar{u}_i, \bar{v}_i)$. Hence $\alpha(p) = A_1 A_2 \ldots A_\ell$ and $\sigma(p) = A$.
(We assume that $1 \leq r_1 < r_2 < \ldots < r_\ell \leq k$).

In order to define Φ_p we need a preliminary construction.

This construction will use the set W(p) of attribute occurrences associated with p and $\underline{Attr}$ as defined above.

For each $i = 0, \ldots, k$, each variable y in $\{x_1, \ldots, x_n, y_1, \ldots, y_q, z_1, \ldots, z_m\}$ we construct a term $t(y,i) \in M(\mathcal{F}, W(p)) \cup \{\perp\}$ representing the value of y at the node s_i on any computation sequence following the path p.

Here is the definition of t :

$t(y,i) = y(0)$ for all y in $\{x_1, \ldots, x_n\}$, all i.

We now assume that $y \in \{y_1, \ldots, y_q, z_1, \ldots, z_m\}$:

$t(y,0) = \perp$, a constant symbol meaning that y is undefined at s_0

$t(y,i+1) = t(y,i)$ if $\theta_{i+1} = \underline{null}$ or θ_i is a guard

Let us now assume that θ_{i+1} is an assignment $\bar{u} \leftarrow (t_1, \ldots, t_r)$ with $\bar{u} = (u_1, \ldots, u_r)$ then $t(y,i+1) = t(y,i)$ if $y \notin \bar{u}$ and $t(y,i+1) = t_j[t(y,i) / y]$ if $y = u_j$

Let us now assume that θ_{i+1} is $\underline{call}$ $B(\bar{u};\bar{v})$ with $\bar{u} = (u_1, \ldots, u_{n'})$ and $\bar{v} = (v_1, \ldots, v_{m'})$.

Then

$t(y,i+1) = t(y,i)$ if $y \notin \bar{v}$ and

$t(y,i+1) = z_j(h)$ if $y = v_j$ and $i+1 = r_h$. (θ_{i+1} is the h-ith procedure call on ; note that $z_j(h)$ is an attribute occurrence).

Note that if y is defined as s_i, then $t(y,i)$ does not contain $\perp$ (more precisely $t(y,i) \in M(\mathscr{F},W(p))$) by induction on i otherwise $t(y,i) = \perp$.

We now define Φ_p as $\mathbb{M}\, C_p \wedge \mathbb{M}\, \Gamma_p$ where C_p and Γ_p are two (finite) set of formulas defined as follows.

For every $i = 1,\ldots,k$ such that θ_i is a guard, say $R(u_1,\ldots,u_r)$, (or not $(R(u_1,\ldots,u_r))$) one puts in C_p the condition $R(t(u_1,i), t(u_2,i),\ldots, t(u_r,i))$ (or not$(R(t(u_1,i), t(u_2,i),\ldots, t(u_r,i))))$.

We now define Γ_p. For defining the synthesized attributes at the root, we put in Γ_p the semantic rules :

$$z_i(0) = t(z_i,k)$$

for all $i=1,\ldots,m$.

For defining the inherited attributes at the successors of the root we put in Γ_p the semantic rules

$$x_i(j) = t(u_i,j)$$

for all $j=1,\ldots,\ell$ (recall that $\alpha(p) = A_1\ A_2\ \ldots\ A_\ell$), all $i=1,\ldots,n$ where the rank of A_j is (n_j,m_j) and u_i is the i-th actual input parameter in the corresponding procedure call, i.e. θ_{r_j}.

Note that these semantic rules assign to formal input parameters the values of the corresponding actual input parameters. This definition makes very clear the parameter passing rule that is considered,and is very close to actual implementation.

Note also that the local variables have disappeared ; they have been replaced by terms denoting their values.

(3.3.3) Example

Consider the system Σ' of example (3.1.4). There corresponds to Σ' the {A,B}-signature

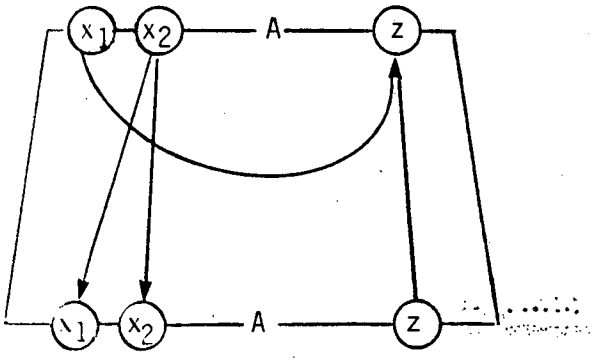
$p_1 : A \to A$

$p_2 : B \to A$

$p_3 : \varepsilon \to B$

$p_4 : AB \to B$

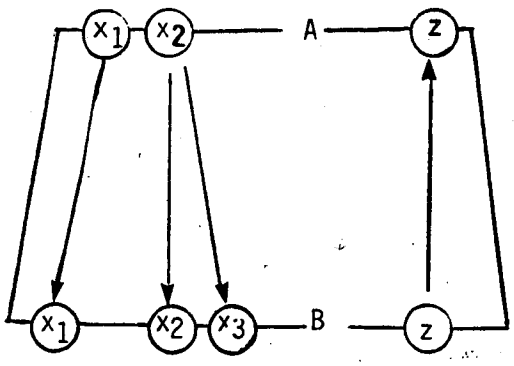
On fig 9., the corresponding semantic rules are shown as usual with the additional convention that if one has an arc $a(i) \to b(j)$ on the graph and no equation with lef handside $b(j)$ then one has, implicitely $b(j) = a(i)$. □



$x_1(1) = h(x_1(0))$

$z(0) = f(x_1(0), z(1))$

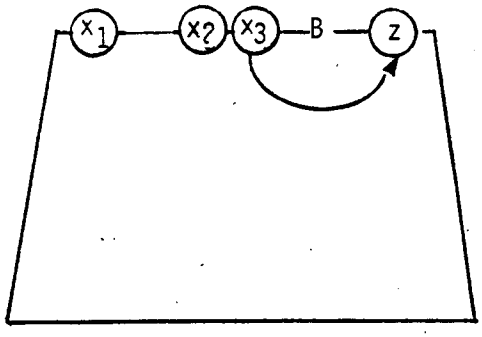
ϕ_{p_1}



$x_2(1) = h(x_1(0))$

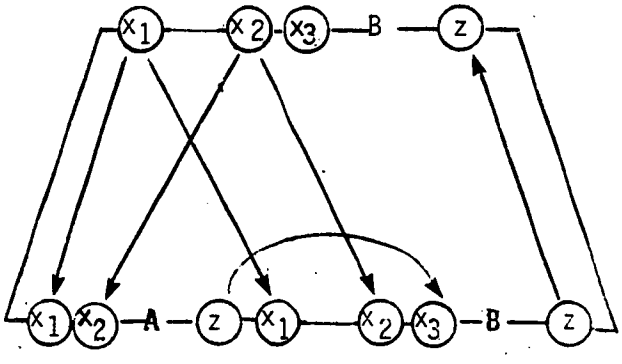
$x_3(1) = g(h(x_1(0)), f(x_1(0), x_2(0)))$

ϕ_{p_2}



$p(x_1(0), x_3(0))$

ϕ_{p_3}



not $(p(x_1(0), x_3(0)))$

ϕ_{p_4}

Fig.9

Let G_Σ be the relational attribute grammar associated in this way with a loop-free system Σ of recursive definitions.

Let D_Σ be the attribute dependency scheme associated with $\{\Gamma_p\}_{p\in P}$ in the usual way.

(3.3.4) Theorem :

Let Σ be a loop-free well-formed system of recursive definitions.

(1) D_Σ is an attribute dependency scheme of type L.

(2) For every A in $\mathcal{A}$ of rank (n,m), for every interpretation I with domain D, for every $\bar{d}$ in D^n and every $\bar{d}' \in D^m$, then $(\bar{d},\bar{d}') \in A_I$ iff there exists a tree t in $M(P)_A$ such that $(\bar{d},\bar{d}') \in R_{t,I}$ (with Attr(A) = $(x_1,\ldots,x_n,z_1,\ldots,z_m)$).

Sketch of the proof :

(1) By an easy inspection of the formal construction.

(2) Observe first that there is a one-to-one correspondence between $M(P)_A$ and the "story" of derivations of the paths $P_A = \bigcup \{Paths_0(T) \mid A \overset{*}{\Rightarrow} T\}$ (c.f. the notations of (3.2)). Roughly speaking <A,P> is the abstract context-free grammar underlying a context-free grammar generating the set P_A from start symbol A. (This is not exactly so due to the renamings and the substitutions of variables at procedure evaluation. See (3.3.5) below).

Let π_A be the corresponding mapping, associating a path with a tree. The result follows then from the

Claim : For all t in $M(P)_A$, all $(\bar{d},\bar{d}')$ in D^{n+m} $(\bar{d},\bar{d}') \in R_{t,I}$ iff $(\bar{d},\bar{d}') \in \pi_A(t)_I$.

This claim can be proved by structural induction on t (simultaneously for all A in $\mathcal{A}$). □

(3.3.5) Some remarks on the semantics of recursive procedures.

The set of paths P_A is not a context-free language. This is due to the renaming of $\bar{y}$ into $\bar{y}'$ and the substitutions $\bar{u}/\bar{x}$ and $\bar{v}/\bar{z}$ which are performed together with the graph substitution in a step $S \to T$ where call $A(\bar{u},\bar{v})$ is rewritten (c.f. (3.2.1)).

The set P_A would have been defined without parameters so as to work by side-effect on a set of global variables.

Rather than formal definitions we give a representative example.

Example.

Let $A = S_A$ be the recursive procedure working on the set $\{x_1, x_2, x_3\}$ of variables where S_A is shown on Fig.10.

The set P_A is generated by the following context-free grammar :

$A \rightarrow \theta_1 \ ; \ \theta_2 \ ; \ \theta_3$

$A \rightarrow \theta_4 \ ; \ A \ ; \ \theta_5$

where $\theta_1 \ldots \theta_5$ stand for instructions

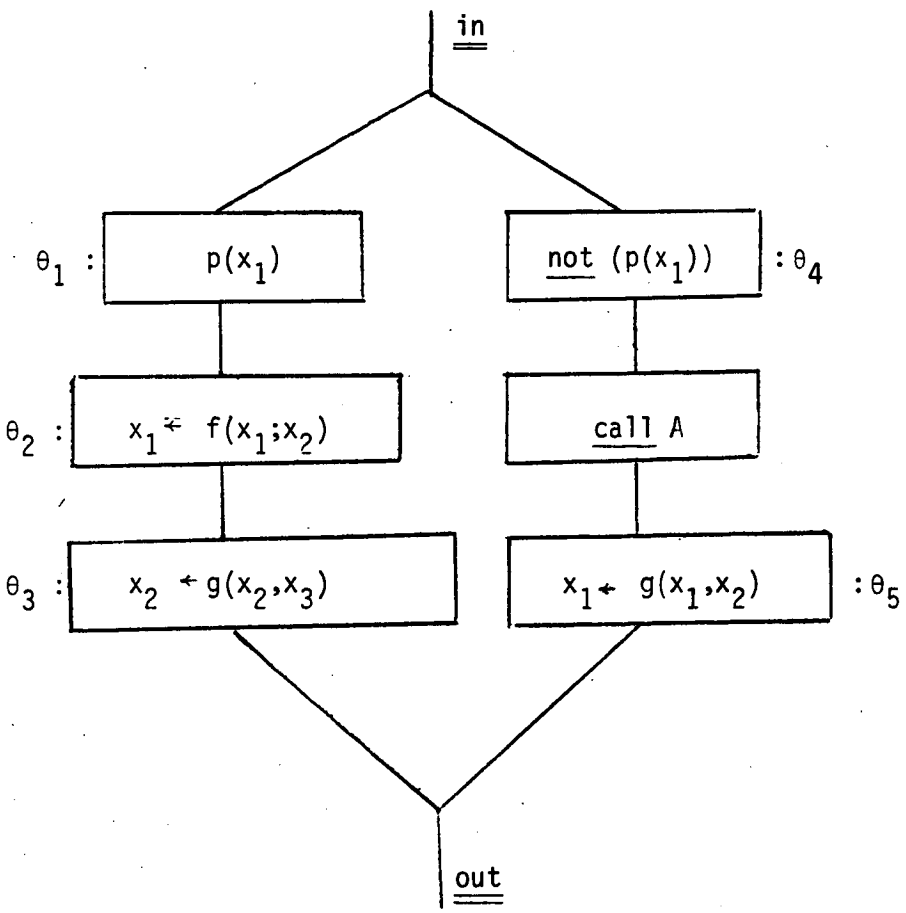


Fig. 10

as shown on Fig.10 and ";" is used, instead of names of nodes (as in (3.2)) to separate instructions.

Hence $P_A = \{\theta_4^n \theta_1 \theta_2 \theta_3 \theta_5^n / n \geq 0\}$.

It is easy to verify that the relation $A_I \subseteq D^3 \times D^3$ computed by A in an interpretation I is $U\{p_I / p \in P_A\}$. □

Let us now go back to our recursive procedures with parameters, local variables and no global variable.

The renamings and substitutions incorporated in the replacement of call $A(\bar{u},\bar{v})$ by S'_A (c.f. (3.2.1)) could be simulated by an infinite set of context-free production rules with non-terminals of the form $(A,\bar{u},\bar{v})$.

Hence P_A is "context-free", in an extended sense where finitely many "production rule schemes" represent an infinite set of production rules.

(3.4) Application to partial correctness.

Let (Σ,A) be a scheme over $(\mathscr{F},\mathscr{R},\mathscr{A})$ of rank (n,m). Let $\phi(\bar{x})$ and $\psi(\bar{x},\bar{z})$ be formulas of some logical calculus $\mathscr{L}$ constructed over $\mathscr{F} \cup \mathscr{R}$.

Let I be an interpretation with domain D. We say that $(\Sigma,A)^\circ$ is partially correct w.r.t. (ϕ,ψ) in I if for all $\bar{d}$ in D^n, all $\bar{d}'$ in D^m,

if $\phi(\bar{d})$ holds, if $(\bar{d},\bar{d}') \in A_I$ then $\psi(\bar{d},\bar{d}')$ holds.

This property is called ∀- partial correctness in [Gal81] where other correctness conditions were considered.

Note that A is partially correct w.r.t. (ϕ,ψ) iff it is w.r.t. $(\text{true},\phi \Rightarrow \psi)$.

Given (Σ,A) and (ϕ,ψ), we define a specification θ for G_Σ as follows :

θ^A : $\phi(\bar{x}) \Rightarrow \psi(\bar{x},\bar{z})$

θ^B : true

for all B in $\mathcal{A}$, $B \neq A$.

For any interpretation I, it is clear from theorem (3.3.4) that (Σ,A) is partially correct w.r.t. (ϕ,ψ) iff G_Σ is valid for θ.

Hence the proof methods derived from theorems (2.2.5) and (2.4.3) can be applied to G_Σ and θ and yield sound and complete methods for estabishing the partial correctness of (Σ,A) w.r.t. (ϕ,ψ).

It is interesting to compare the inductive specification θ_G of theorem (2.2.5) with the inductive assertions defined by Gallier [Gal 81] to establish the completeness of his method.

We do it on an example.

(3.4.1) Example

Let $A(x;z) = S_A$ be a recursive definition such that S_A is the flowchart shown on Fig.11. The interpretation is the domain of integers with the usual operations.

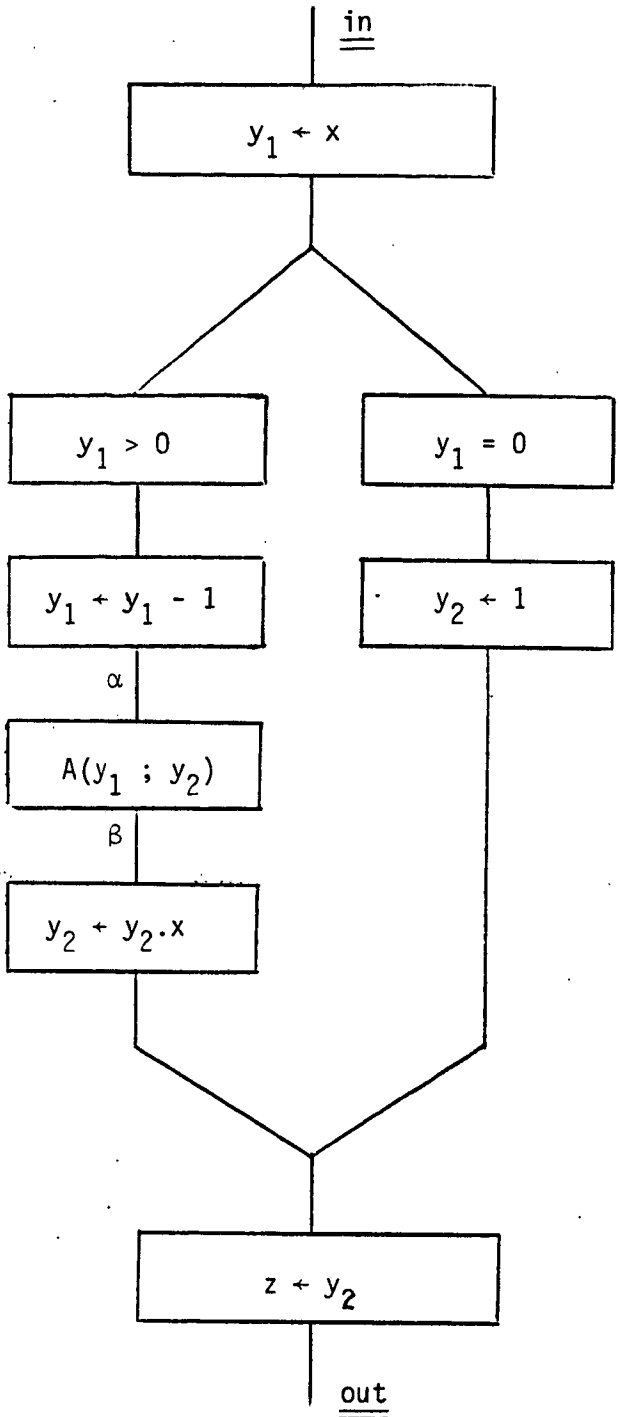

in
$y_1 \leftarrow x$
$y_1 > 0$
$y_1 = 0$
$y_1 \leftarrow y_1 - 1$
$y_2 \leftarrow 1$
α
$A(y_1 ; y_2)$
β
$y_2 \leftarrow y_2.x$
$z \leftarrow y_2$
out

Fig.11 A

We shall prove the partial correctness of A with respect to $(\phi(x),\psi(x,z))$ where

$\phi(x)$: $x \geq 0$

$\psi(x,z)$: $z \geq 1$ and $\lceil (x+1)/2 \rceil$ divides z, (where $\lceil y \rceil$ denotes the least integer y' such that $y' \geq y$).

Since A(x;z) defines $z = x!$ whenever $x \geq 0$ its partial correctness follows from an easy arithmetical argument. Gallier's method consists in finding predicates $\phi_A(x)$, $\psi_A(x,z)$ and $q_\alpha(x,y_1)$ such that, for all x,y_1,y_2 :

(1) $\phi(x) \Rightarrow \phi_A(x)$

(2) $\phi_A(x)$ and $x = 0 \Rightarrow \psi_A(x,1)$

(3) $\phi_A(x)$ and $x > 0 \Rightarrow q_\alpha(x,x-1)$ and $\phi_A(x-1)$

(4) $q_\alpha(x,y_1)$ and $\psi_A(y_1,y_2) \Rightarrow \psi_A(x,y_2.x)$

(5) $\phi(x)$ and $\psi_A(x,z) \Rightarrow \psi(x,z)$.

Intuitively ϕ_A is an input predicate and ψ_A is an input-output predicate for A such that A is partially correct w.r.t. ϕ_A, ψ_A. The predicate $q_\alpha(x,y_1)$ is a relation that is satisfied for all computation from in to α in S_A (by (3)) and is sufficient to insure the validity (with respect to ψ_A) of every computation in S_A from α to out. (By (4)).

Conditions (1) and (5) say that (ϕ_A,ψ_A) is stronger than (ϕ,ψ).

Condition (2) states the correctness w.r.t (ϕ_A,ψ_A) of the computations of S_A following the right most path (see Fig.10).

Conditions (3) and (4) state the same thing for the computations following the left most path, if one assumes the correctness of the call to A on this path, and by using q as an intermediate predicate.

Note that q_α can be eliminated between (3) and (4) i.e. that the conjunction of (3') and (4') is sufficient to establish the partial correctness of A w.r.t.(ϕ,ψ) :

(3') $\phi_A(x)$ and $x > 0 \Rightarrow \phi_A(x-1)$

(4') $\phi_A(x)$ and $x > 0$ and $\psi_A(x-1,y_2) \Rightarrow \psi_A(x,y_2.x)$

The completeness theorem of [Gal 81] shows that one can take for ϕ_A the predicate true and for ψ_A the input-output relation defined by A i.e. $\psi_A(x,z)$ iff $x \geq 0$ and $z = x$!.

The following formula being false (in $\mathbb{N}$) :

(*) $\forall x,y\ [x,y > 0$ and $\lceil x/2 \rceil$ divides $y \Rightarrow \lceil (x+1)/2 \rceil$ divides $x.y]$

(it suffices to take $x = 12$, $y = 6$) one cannot simply take ψ instead of ψ_A. This is a example of the frequent situation where for doing a proof by induction, one needs an inductive assertion which is stronger than the one to be established.

Let us now do this proof of partial correctness in terms of attribute grammars. The relational attribute grammar associated with A consists of two "productions". The first one is p_1 with profile <A,A>. The associated relation Φ_{p_1} is :

$x(0) > 0$ and $x(1) = x(0)-1$ and $z(0) = z(1).x(0)$.

The second one is p_2 of profile $<\varepsilon,A>$ with relation Φ_{p_2} :

$x(0) = 0$ and $z(0) = 1$

The specification (ϕ,ψ) is now written as a unique formula $\phi \Rightarrow \psi$ that we denote by θ and is :

$x \geq 0 \Rightarrow z \geq 1$ and $\lceil (x+1)/2 \rceil$ divides z

Since (*) is false in $\mathbb{N}$ so is

Φ_{p_1} and $\theta[x(1)/x, z(1)/z] \Rightarrow \theta[x(0)/x, z(0)/z]$

hence θ is not inductive.

The completeness theorem (2.2.5) shows that in order to prove the validity of the relational attribute w.r.t. θ (hence the partial correctness of A w.r.t. (ϕ,ψ)), it suffices to prove that $\theta_G \Rightarrow \theta$ where θ_G is the strongest valid specification, which is nothing else by theorem (3.3.4) than the input-output relation of A.

It follows that θ_G coincides with $\phi_A \Rightarrow \psi_A$ where ϕ_A, ψ_A follow from Gallier's completeness proof.

(4) Application to recursive applicative procedures

Whereas the operational semantics of recursive imperative is unique, it is well known that for recursive applicative procedures there are several evaluation strategies (call-by-name, call-by-value, left-most outer-most, etc...) yielding possibly different results. These facts have been investigated by Cadiou [Cad75], Vuillemin [Vui74], Downey and Sethi [DS76] among others.

We introduce a class of recursive applicative procedures (defined in terms of program schemes), a call-by-value and a call-by-name computation rule, and for each of them the construction of a relational attribute grammar representing the corresponding semantics. Proof methods for establishing the partial validity follow immediately and yield completeness results which are new to the authors' knowledge.

(4.1) Recursive applicative procedures and their computation rules.

(4.1.1) Definition (Recursive applicative procedures, interpretation).

Let $\mathscr{F}$ and $\mathscr{R}$ be as in section 3.

Let Ψ be a finite set of function symbols with arity (ψ in Ψ has arity $\rho(\psi) \geq 1$

A system of recursive applicative procedures is an object $\Sigma = \langle \psi(x_1,\ldots,x_{\rho(\psi)}) \Leftarrow s_\psi ; \psi \in \Psi \rangle$ where for each ψ in Ψ , s_ψ is a conditional expression i.e. an expression of the form $(c_1 \rightarrow s_1, c_2 \rightarrow s_2, \ldots, c_n \rightarrow s_n)$ where $c_1,\ldots,c_n$ are Boolean combinations of elements of $\mathscr{R}(X_k)$ (we let $k = \rho(\psi)$) and $s_1,\ldots,s_k \in M(\mathscr{F} \cup \Psi, X_k)$. A longer but more intuitive notation for s_ψ would be :

$\underline{\text{if}}\ c_1\ \underline{\text{then}}\ s_1\ \underline{\text{elseif}}\ c_2\ \underline{\text{then}}\ s_2\ \underline{\text{elseif}}\ \ldots\ \underline{\text{elseif}}\ c_n\ \underline{\text{then}}\ s_n$

Note that we are defining deterministic recursive procedures.

An interpretation for Σ is an object I exactly as in section (3). □

(4.1.2) Definitions (computation as a sequence of rewritings)

There exist many computation rules (call-by-value, call-by-name) all of them expressed as specific ways of applying rewritings in a certain rewriting system working on expressions, Boolean expressions, terms in $M(\mathcal{F} \cup \Psi, X))$ where values from D_I have been substituted for variables.

The basic steps of this rewriting system are the following ones :

I For conditional expressions :

$(\underline{\text{true}} \to s_1,\ c_2 \to s_2, \ldots,\ c_n \to s_n) \to s_1$

$(\underline{\text{false}} \to s_1,\ c_2 \to s_2, \ldots,\ c_n \to s_n) \to (c_2 \to s_2, \ldots,\ c_n \to s_n)$

II For Boolean expressions :

$\underline{\text{true}}\ \underline{\text{and}}\ e_1 \to e_1$
$\underline{\text{false}}\ \underline{\text{and}}\ e_1 \to \underline{\text{false}}$

$\underline{\text{true}}\ \underline{\text{or}}\ e_1 \to \underline{\text{true}}$
$\underline{\text{false}}\ \underline{\text{or}}\ e_1 \to e_1$

$\underline{\text{not}}\ (\underline{\text{true}}) \to \underline{\text{false}}$
$\underline{\text{not}}(\underline{\text{false}}) \to \underline{\text{true}}$

Hence we have introduced sequential and and or. One could also add the rules e_1 and false → false and e_1 or true → true (defining parallel and and or, c.f. Vuillemin [Vui75] but we do not here.

III For terms with leading function symbol in $\mathscr{F} \cup \mathscr{R}$,

$$f(d_1,\ldots,d_k) \to d$$

if $f \in \mathscr{F} \cup \mathscr{R}$, $k = \rho(f)$, $d_1,\ldots,d_k \in D_I$

$$d = f_I(d_1,\ldots,d_k)$$ (note that d is true or false if $f \in \mathscr{R}$)

IV For terms with recursively defined leading function symbol :

$$\psi(e_1,\ldots,e_k) \to s_\psi[e_1/x_1,\ldots,e_k/x_k]$$

if $\psi \in \Psi$, $k = \rho(\psi)$, $e_1,\ldots, e_k \in M(\mathscr{F} \cup \Psi, D_I)$

Remark : All rules of groups I to III reduce the size of the term to which they apply. Hence they necessarly terminate.

As usual a basic step can be performed on a subexpression of some expression. We shall denote by $\overset{*}{\to}$ the transitive and reflexive closure of the rewriting relation thus associated with all rules of groups I to IV.

It is well-known (see Vuillemin [Vui 75], Cadiou [Cad 75]) that there exists at most one element d if D_I such that

$$\psi(d_1,\ldots,d_k) \overset{*}{\to} d$$

and we shall denote it as $\psi_I(d_1,\ldots,d_k)$. This notation defines ψ_I as a partial function $D_I^k \to D_I$. It is the function computed by ψ in I.

For every expression e, several basic steps of computation can be applied.

A computation rule is a way to select for each expression e which of all possible computation steps will be applied.

Hence every computation rule $\mathscr{C}$ defines a restriction $\overset{*}{\to}_{\mathscr{C}}$ of $\overset{*}{\to}$. It follows that the function defined by ψ in I with $\mathscr{C}$ i.e. $\psi_{I,\mathscr{C}}$ such that

$$\psi_{I,\mathscr{C}}(d_1,\ldots,d_k) = d \quad \text{iff } \psi(d_1,\ldots,d_k) \overset{*}{\to}_{\mathscr{C}} d$$

is a restriction of ψ_I, i.e. $\psi_{I,\mathscr{C}} \subseteq \psi_I$, and the inequality may be strict.

We shall introduce two specific computation rules, the call-by-value and the call-by-name. Other ones can be found in Vuillemin [Vui 74].

(4.1.3) Definition : Call-by-value.

This computation rule can be specified as follows, by saying how to evaluate an expression :

(1) For every conditional expression of the form

$$(c_1 \to s_1,\ c_2 \to s_2,\ \ldots,\ c_n \to s_n)$$

evaluate c_1 into c_1' and then apply a rule of type I to $(c_1' \to s_1, c_2 \to s_2, \ldots, c_n \to s_n)$.

(2) For a Boolean expression $c[e_1/x_1,\ldots,e_k/x_k]$ where c is a Boolean combination of elements of $\mathscr{R}(X_k)$ such that $x_1,\ldots,x_k$ all occur in c and $e_1,\ldots,e_k \in M(\mathscr{F} \cup \Psi, D_I)$, evaluate $e_1,\ldots,e_k$ into $d_1,\ldots,d_k$ and then apply rules of type II and III to $c[d_1/x_1,\ldots,d_k/x_k]$.

(3) For a term of the form $f(e_1,\ldots,e_k)$ with $f \in \mathscr{F}_k$, $e_1,\ldots,e_k \in M(F \cup \Psi, D_I)$, evaluate $e_1,\ldots,e_k$ into $d_1,\ldots,d_k$ and then apply a rule of type III to $f(d_1,\ldots,d_k)$.

(4) For a term of the form $\psi(e_1,\ldots,e_k)$ with $\psi \in \Psi_k, e_1,\ldots,e_k \in M(F \cup \Psi, D_I)$ evaluate $e_1,\ldots,e_k$ into $d_1,\ldots,d_k$ and then apply a rule of type IV to $\psi(d_1,\ldots,d_k)$.

This is a (partly informal) recursive definition of a procedure evaluate which defines an interpretor for this "programming language". It is very close to the classical LISP interpretor (Mc Carty et al. [MC68]).

We shall denote by $\psi_{I,val}$ the mapping defined by ψ in I with this computation rule, called the call-by-value.

Examples can be given where $\psi_{I,val} \neq \psi_I$. (See example (4.3) below).

(4.1.4) Definition : Call-by-name.

It can be specified in a similar way as follows :

(1') as (1) for call-by-value.

(2') For a Boolean expression of the form not(e_1), e_1 and e_2, e_1 or e_2 evaluat e_1 first and then apply a rule of type II.

For a Boolean expression of the form $\mathscr{R}(e_1,\ldots, e_k)$ for $\mathscr{R} \in \mathscr{R}_k$, evaluate $e_1,\ldots,e_k$ and then apply a rule of type III.

(3') as (3) for call-by-value.

(4') For a term of the form $e = \psi(e_1,\ldots,e_k)$ with $\psi \in \Psi_k$, $e_1,\ldots,e_k \in M(F \cup \Psi, D_I$ use a rule of type IV at the left most position i.e. replace e by $s_\psi[e_1/x_1,\ldots,e_k/x$

This computation rule is called the call-by-name, and defines a function $\psi_{I,\text{name}}$.

It can be shown that $\psi_{I,\text{name}} = \psi_I$. Vuillemin [Vui75].

(4.1.5) Remark :

Using rules (1), (2'), (3) and (4) would give another computation rule $\mathscr{C}$, such that

$$\psi_{I,\text{val}} \subseteq \psi_{I,\mathscr{C}} \subseteq \psi_{I,\text{name}}$$

with possibly strict inclusions.

(4.2) Defining $\psi_{I,\text{val}}$ and $\psi_{I,\text{name}}$ by attributes

(4.2.1) A relational attribute grammar to compute $\psi_{I,\text{val}}$.

Let Σ be as in (4.1.1).

We defined an abstract context-free grammar $\langle \Psi, P \rangle$ where Ψ is the set of procedure symbols of Σ and P is defined as follows.

For every ψ in Ψ with definition

$$s_\psi = (c_1 \rightarrow s_1,\ c_2 \rightarrow s_2,\ \ldots,\ c_n \rightarrow s_n)$$

for every i in {1,...,n} we introduce a symbol $p_{\psi,i}$ in P of sort ψ and of arity $\psi_1\ \psi_2 \ldots \psi_k$ where $\psi_1, \psi_2, \ldots, \psi_k$ is the sequence of symbols in Ψ appearing in s_i in this order when s_i is written linearly in Polish postfix notation.

Hence P is a Ψ-signature.

Let us associate with $c_1, \ldots, c_n$ in s_ψ the following logical formulas :

For i = 1,...,n we define c_i' to be the logical formula expressing that the i-th case of the conditional expression holds, namely :

$$c_1' = c_1$$

$$c_2' = \underline{\text{not}}(c_1)\ \ \underline{\text{and}}\ c_2$$

$$c_n' = \underline{\text{not}}(c_1)\ \ \underline{\text{and}}\ \ \underline{\text{not}}(c_2)\ \underline{\text{and}}\ \ldots\ \underline{\text{not}}(c_{n-2})\ \ \underline{\text{and}}\ \ c_n$$

This precise writing of c_i' is relevant since we shall use a "sequential" non commutative and-operator.

In order to simplify the notations, for $p = p_{i,\psi}$ in P, it will be convenient to denote by s_p the term s_i of s_ψ and by c_p' the logical formula c_i' (also defined w.r.t. s_ψ ; see above).

We now proceed to the definition of our relational attribute grammar.

Although this is not necessary in a relational attribute grammar, the set $\underline{\text{Attr}}(\psi)$ of attributes of ψ will be splitted into $\underline{\text{Inh}}(\psi) \cup \underline{\text{Syn}}(\psi)$ where if $\rho(\psi)=k$, we let $\underline{\text{Inh}}(\psi) = \{x_1, \ldots, x_k\}$ and $\underline{\text{Syn}}(\psi) = \{z_\psi\}$ (this attribute will define the value of ψ for input values held by the inherited attributes).

The definition of Φ_p will use an auxiliary mapping denoted by Dec (for decomposition).

Let $t \in M(\mathcal{F} \cup \Psi, X_k)$; let $w_1, w_2, \ldots, w_\ell$ be the list of all nodes of t (viewed as a tree) where a symbol of Ψ occurs ; the nodes are formally defined as Dewey words and they are listed in post-order i.e. in the order corresponding to writing t in Polish postfix notation.

For every node u of t, we define a term Dec(t,u) as follows :

Dec(t,u) = a if u is a leaf labeled by a in $\mathcal{F}_0$,

Dec(t,u) = x(0) if u is a leaf labeled by x in X_k,

Dec(t,u) = f(Dec(t,u1),...,Dec(t,uh)) if u is labeled by f in $\mathcal{F}_h$, $h \geq 1$,

Dec(t,u) = $z_\psi(i)$ if $u = w_i$, $1 \leq i \leq \ell$ and u is labeled by ψ.

Note that $Dec(t,u) \in M(F,W)$ where $W \subset \{a(i)/i \in \mathbb{N}, a \in Attr\}$ i.e. can be considered as a set of attribute occurences.

For $p \in P$, we now define Φ_p as the conjunction of a Boolean condition B_p together with a set Γ_p of semantic rules defining some attributes in terms of other ones as in ordinary attribute grammars.

We let B_p be the formula

$$c'_p[x_1(0)/x_1, \ldots, x_k(0)/x_k]$$

We now define Γ_p as a set of equations, called semantic rules. We first put in Γ_p the semantic rule

$$z_\psi(0) = Dec(s_p, 0)$$

which defines z_ψ at the root of p.

We need the semantic rules defining the inherited attributes (corresponding to the formal input parameters of the procedure symbols appearing in s_p), and we put

in Γ_p the rules

$$x_i(j) = \underline{Dec}(t_p, w_j i)$$

for all $1 \leq j \leq \ell$, all $1 \leq i \leq \rho(\psi_j)$ where ψ_j is the procedure symbol occurring at w_j and $w_1,\ldots,w_\ell$ is the list of nodes in s_p where some symbol of Ψ occurs (in post-order).

Hence we have defined a relational attribute grammar $G_{\Sigma,I} = \langle \Psi, P, \Phi, I \rangle$ where I is any interpretation for Σ.

With these notations we have :

(4.2.2) Theorem :

If $\psi \in \Psi_k$, *if* $\bar{d} \in D_I^k$ *then, for all* d' *in* D_I , $\psi_{I,\underline{val}}(\bar{d}) = d'$ *iff there exists* $t \in M(P)_\psi$ *such that* $(\bar{d},d') \in R_{t,I}$.

The proof of this theorem will be based on the following lemma, stated with the notations of the definition of Φ_p :

(4.2.3) Lemma :

Let $(\bar{\psi})_{\psi \in \Psi}$ *be a family of functions,* $\bar{\psi} : D_I^{\rho(\psi)} \to D_I$. *Let* $p \in P$, *let* $d_1,\ldots,d_k \in D_I$, *let* $d' = \alpha(s_p)$ *where for* $t \in M(\Psi \cup \mathscr{F}, X_k)$, $\alpha(t)$ *denote the value of* $t[d_1/x_1,\ldots,d_k/x_k]$ *computed in* I *with* $\bar{\psi}$ *interpreting each symbol* ψ *of* Ψ.

Let $\nu : W(p) \to D_I$ *be the assignment such that* $\nu(z_\psi(0)) = d'$, $\nu(x_i(0)) = d_i$ *for* $i = 1,\ldots,k$, $\nu(x_i(j)) = \alpha(s_p/w_j i)$ *and* $\nu(z_{\psi_j}(j)) = \alpha(s_p/w_j)$ *then* ν *satisfies all the equations of* Γ_p.

Proof : It can be shown by induction on u that :

$$\alpha(t/u) = \underline{Dec}(t,u)_I[\nu(w)/w \ ; \ w \in W(p)]$$

for all t and u as in the definition of $\underline{Dec}(t,u)$.

The result follows then from the definition of Γ_p. □

Sketch of the proof of theorem (4.2.2) :

Let $\psi_{I,val}(\bar{d}) = d'$. We construct a tree of calls t in $M(P)_{\psi}$ and an assignment $\nu : W(t) \to D$ which satisfies Φ_t and such that $\nu(x_i(0)) = d_i$, $\nu(z_{\psi}(0)) = d'$, by using an induction on the length m of a call-by-value computation γ :

$\psi(\bar{d}) \overset{*}{\to} d'$.

We first define $\nu(x_i(0)) = d_i$ and $\nu(z_{\psi}(0)) = d'$.

Let $s_{\psi} = (c_1 \to s_1, c_2 \to s_2, \ldots, c_n \to s_n)$. The first steps of γ consist in selecting some $i \in \{1,\ldots,n\}$ such that c_i is true for $\bar{d}$. There is a unique one say i_o and let $p = p_{i_o,\psi}$. Hence ν satisfies B_p.

If s_p has no occurrence of symbols in Ψ, then p is of arity ε, we let $t = p$ and since $s_p[\bar{d}]$ evaluates to d' in I., ν satisfies Γ_t which is reduced to the single equation

$$z_{\psi}(0) = s_p[x_i(0)/x_i,\ 1 \le i \le \rho(\psi)]$$

We now assume that $\alpha(p) = \psi_1\ \psi_2 \cdots\ \psi_{\ell}$.

Then the remaining of γ, namely $\gamma' : s_{i_o}[\bar{d}] \overset{*}{\to} d'$ consists of computations $\gamma_i : \psi_i[\bar{d}^{(i)}] \overset{*}{\to} d'^{(i)}$ for all $i = 1,\ldots,\ell$.

By the post order enumeration of $w_1,\ldots,w_{\ell}$ the computation γ_i does not use the result of any $\gamma_{i'}$ for any $i' > i$.

By induction hypothesis, there exists trees $t_1,\ldots,t_{\ell}$ in $M(\Psi)_{\psi_1},\ldots, M(\Psi)_{\psi_{\ell}}$ and assignments $\nu_1,\ldots,\nu_{\ell}$ respectively associated with $\gamma_1,\ldots,\gamma_{\ell}$.

We let $t = p(t_1,\ldots,t_{\ell})$ and $\nu(a(iw)) = \nu_i(a(w))$ for all $i=1,\ldots,\ell$, all $a(w) \in W(t_i)$

By taking $(\bar{\psi})_{\psi \in \Psi}$ such that $\bar{\psi} = \psi_{I,val}$ extended into a total function in an arbitrary way if necessary, we can apply lemma (4.2.3) which shows that ν satisfies the equations of Γ_p.

Finally ν satisfies Φ_t (since by induction ν_i satisfies Φ_{t_i}).

Conversely, let $t \in M(\Psi)_\psi$, let ν be a valid t-assignment, let $\bar{d} = (\nu(x_1(0)),\ldots, \nu(x_k(0)))$ and $d' = \nu(z_\psi(0))$. We must show that $\psi_{I,\underline{val}}(\bar{d}) = d'$.

The proof is an induction on the structure of t, and consists in reversing the one of the first part. □

Hence we have shown how to represent by a relational attribute grammar the call-by-value semantics of our recursive procedures.

By inspecting the definition of Γ_p one can establish

(4.2.4) Proposition :

The attribute dependency scheme associated with the sets of semantic rules $(\Gamma_p)_{p\in P}$ *of* $G_{\Sigma,I}$ *is of type* L.

Applications to the proof of partial correctness follow immediately. Let us give an example.

(4.2.5) Example.

Let Σ consist of the following equation.

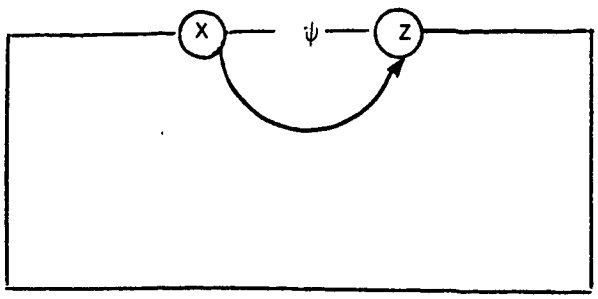
$\psi(x)$ = if $q(x)$ then $f(x,a)$ else $f(\psi(\psi(f(x,b))),c)$

For the interpretation I with domain $\mathbb{Z}$, $q_I(x)$ equivalent to $x > 100$, f_I being the addition, $a_I = -10$, $b_I = 11$, $c_I = 0$, one obtains a classical example of Mc Carthy (see Vuillemin [Vui 75]:

$\psi(x)$ = if $x > 100$ then $x - 10$ else $\psi(\psi(x + 11))$.

The attribute grammar $G_{\Sigma,I}$ will use two "productions" p_1 corresponding to $f(x,a)$ and p_2 corresponding to $f(\psi(\psi(f(x,b),c)$.

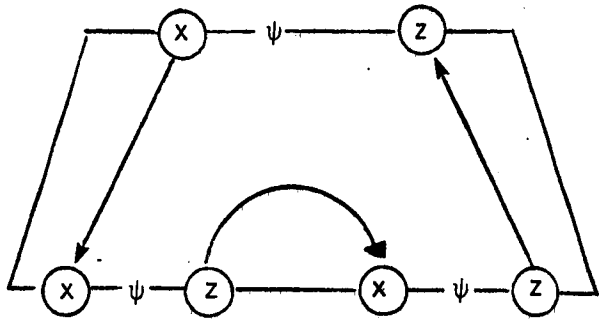
By using x instead of x_1 and z instead of z_ψ we get Φ_{p_1} and Φ_{p_2} shown on Fig.12.



q(x(0))

z(0) = f(x(0),a)

Production p_1



not(q(x(0))

x(1) = f(x(0),b)

z(0) = f(z(2),c)

Production p_2

Fig. 12

The function $\psi_{I,val}$ is such that :

$\psi_{I,val}(x)$ = if x > 100 then x-10 else if x ≤ 100 then 91

To prove this fact we associate with $G_{\Sigma,I}$ the specification θ(x,z) :

(x ≤ 100 and z = 91) or (x > 100 and z = x-10)

Let us verify that θ is inductive.

The condition corresponding to p_2 is obvious. The one corresponding to p_1 is :

$x_0 \leq 100$ and $x_1 = x_0 + 11$ and $x_2 = z_1$ and $z_0 = z_2$

and $[(x_1 \leq 100$ and $z_1 = 91)$ or $(x_1 > 100$ and $z_1 = x_1\text{-}10)]$

and $[(x_2 \leq 100$ and $z_2 = 91)$ or $(x_2 > 100$ and $z_2 = x_2\text{-}10)]$

=> $(x_0 \leq 100$ and $z_0 = 91)$ or $(x_0 > 100$ and $z_0 = x_0\text{-}10)$.

This formula can be easily verified. This verification needs to consider the following cases

1) $x_0 \leq 89$

2) $89 < x_0 \leq 99$

3) $x_0 = 100$. □

(4.2.6) Remark.

Theorem (4.2.2) and proposition (4.2.4) could have be obtained as a corollary of theorem (3.3.4) via a translation of a system of recursive applicative procedures into a system of imperative ones.

We do not define it formally, but we only indicate the general idea and present an example.

Every ψ in Ψ_k will be replaced by a procedure $A_\psi(x_1,\ldots,x_k ; z_\psi)$ of rank(k,1).

By introducing some local variables, the definition s_ψ of ψ can be translated into a loop-free flowchart S_ψ. This translation is based on that of term written in Polish post-fix notation into a sequence of assignments, which is rather obvious.

It is interesting to note that the relational attribute grammar $G_{\Sigma,I}$ associated with a system Σ of recursive applicative procedures coincides with the one associated with the translation of Σ into a system of imperative ones.

If we apply this method to the recursive definition of example (4.2.5), we get for S_ψ the flow-chart of figure 13. □

(4.2.7) Remark.

The construction of $G_{\Sigma,I}$ associated with Σ is essentially the same as that of [DPSS 77] associating an L-attribute grammar with an IO-macro grammar, also used in [EF 81] for a similar purpose.

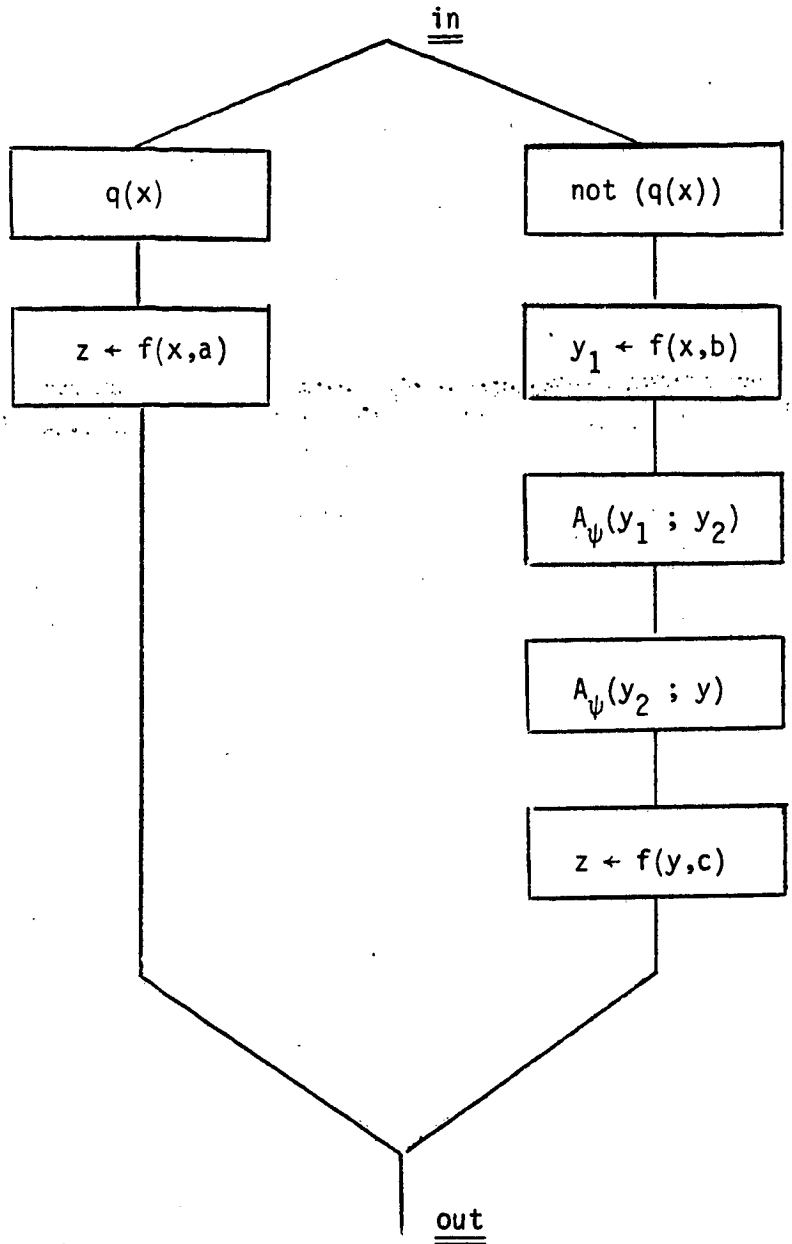


Fig.13 $A_\psi(x;z)$

(4.3) The case of $\psi_{I,name}$.

Let us first recall that $\psi_{I,name} = \psi_I$ and that $\psi_{I,val}$ can be strictly smaller than ψ_I by means of the following example (Cadiou [Cad75]):

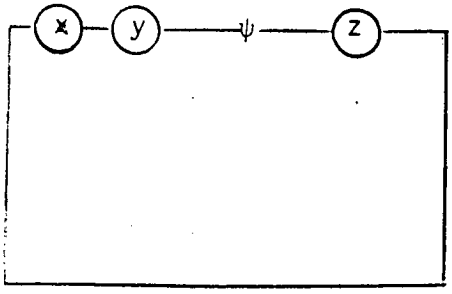
$\psi(x,y)$ = if x = 0 then 0 else $\psi(x-1, \psi(x,y))$

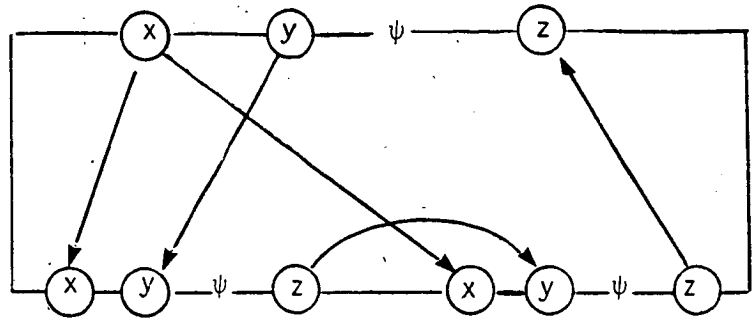
whereas $\psi_{I,name}(1,0) = 0$, $\psi_{I,val}(1,0)$ is undefined due to the sequence of recursive calls :

$\psi(1,0) \rightarrow \psi(0,\psi(1,0)) \rightarrow \psi(0, \psi(0,\psi(1,0)))$

$\rightarrow \ldots$

Let us consider the relational attribute grammar associated with this recursive definition by the construction of (4.2.1). It is shown on figure 14.





$x(0) = 0$ (condition)

$z(0) = 0$

production p_1

$x(0) > 0$ (condition)

$x(2) = x(0) - 1$

production p_2

Fig.14

The call-by-name computation of $\psi_I(1,0)$ into 0 can be represented by the "partial" tree with undefined ($\perp$) attribute occurences z(1) and y(2) of figure 15. This is not a tree in $M(\{p_1,p_2\})$, it corresponds to an incomplete derivation, due to the fact that $\psi(0,\psi(1,0))$ evaluates to 0 without needing to evaluate the inner call (to $\psi(1,0)$).

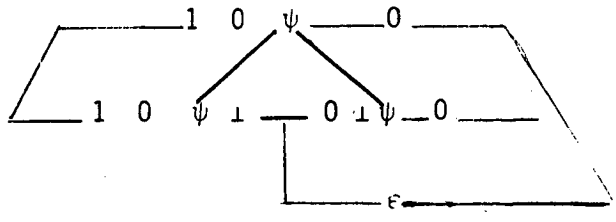


Fig.15

By using the classical Ω-trick (or $\perp$-trick) used for recursive applicative program schemes.

(4.3.1) Construction of a relational attribute grammar modeling call-by-name computat

Let Σ be a system and let P be associated with Σ as in (4.2.1).

Let $P_\Omega = P \cup \{\Omega_\psi \mid \psi \in \Psi\}$ where Ω_ψ is a new symbol of arity ε and of sort ψ.

We introduce a new constant $\perp$ and for every interpretation defined w.r.t. $(\mathscr{F}, \mathscr{R})$ we define $I_\perp$ with respect to $(\mathscr{F} \cup \{\perp\}, \mathscr{R})$:

$D_{I_\perp} = D_I \cup \{\perp\}$

$\perp_I = \perp$ (hence we do not distinguish between the constant symbol $\perp$ and its value in $I_\perp$)

$f_{I_\perp}(d_1,\ldots d_n) = f_I(d_1,\ldots,d_n)$ if $d_1,\ldots,d_n \in D_I$

$= \perp$ if any of $d_1,\ldots,d_n$ is $\perp$

for $f \in \mathscr{F} \cup \mathscr{R}$

Note that for $\mathscr{R}$ in $\mathscr{R}_n$, $\mathscr{R}_{I_\perp}$ is a mapping : $D_{I_\perp}^n \to \{\underline{\text{true}}, \underline{\text{false}}, \perp\}$.

The Boolean operators will satisfy (together with the usual laws) the following one

$\underline{\text{not}}(\perp) = \perp$

$\perp \underline{\text{and}}\ x = \perp$, $\underline{\text{true}}\ \underline{\text{and}}\ \perp = \perp$, $\underline{\text{false}}\ \underline{\text{and}}\ \perp = \underline{\text{false}}$

$\perp \underline{\text{or}}\ x = \perp$, $\underline{\text{true}}\ \underline{\text{or}}\ \perp = \underline{\text{true}}$, $\underline{\text{false}}\ \underline{\text{or}}\ \perp = \perp$

for all x in $\{\underline{\text{true}}, \underline{\text{false}}, \perp\}$.

For $p = \Omega_\psi$ we let Φ_p be the unique condition :

$z_\psi(0) = \perp$

and for p in P, Φ_p is the conjunction the equations of Γ_p (as in (4.2.1)) together with the condition B'_p :

$$c'_p\,[x_1(0)/x_1,\ldots,x_k(0)/x_k] = \underline{\text{true}}$$

which holds iff the left hand -side evaluates to true and neither to false nor to $\perp$.

We shall denote by $G^{\Omega}_{\Sigma,I}$ this relational attribute grammar. The idea of its construction is that the incomplete tree of Fig.14 becomes a complete tree in $M(P_\Omega)$ shown on Fig.16 and all its attributes get a value, possibly $\perp$:

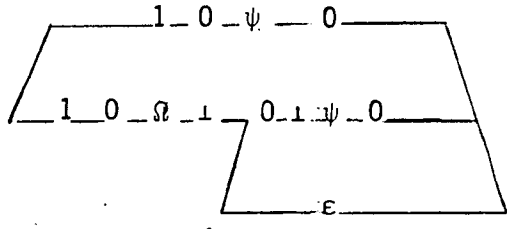


Fig. 16

The proof that $G^{\Omega}_{\Sigma,I}$ defines $\psi_{I,\underline{\text{name}}}$ will require some lemmas.

The following lemma is an immediate consequence of definition (4.1.4), although a thorough proof would be lengthy.

(4.3.2) Lemma :

Let $\psi \in \Psi_k$, $p = p_{\psi,i} \in P$ *(see construction (4.2.1)),*
let $s_\psi = (c_1 \to s_1,\ c_2 \to s_2,\ldots,\ c_i \to s_i,\ldots,\ c_n \to s_n)$, *let* $t = s_\psi[e_1/x_1,\ldots,e_k/x_k]$ *where* $e_1,\ldots,e_k$ *are in* $M(F \cup \Psi, D_I)$. *We shall abbreviate this into* $t = s_\psi[\bar{e}]$.

Let $\gamma : t \overset{*}{\to} d$ *be a call-by-name computation of* t *with* $d \in D_I$.

It is necessarly of the form $\gamma';\gamma''$ *(we denote by ";" the concatenation of computation sequences),*

where $\gamma' : s_\psi[\bar{e}] \overset{*}{\to} s_i[\bar{e}]$ *for some* $i = 1,\ldots,n$ *and* $\gamma'' : s_i[\bar{e}] \overset{*}{\to} d$. *And* $\gamma' = \gamma_1 ; \gamma_2 ; \ldots ; \gamma_i$ *where* γ_j *evaluates* c_j, $j = 1,\ldots,i$ *("evaluates" is taken in the sense of definition (4.1.4)). Along* γ', *some of* $e_1,\ldots,e_k$ *are evaluated getting*

values in D_I, *others are not. Let* $d_{i'}$ *be this value in the former case and* $d_{i'} = \perp$ *in the latter* ($i' = 1,\ldots,k$). *Let* $\bar{d} = (d_1,\ldots,d_k)$.

Then

$c_j[\bar{d}]$ = *false for all* $j = 1,\ldots,i-1$

$c_i[\bar{d}]$ = *true*

$c'_i[\bar{d}]$ = *true.*

The following lemma is an adaptation of lemma (4.2.3) to the call-by-name case.

(4.3.3) Lemma :

Let $(\bar{\psi})_{\psi\in\Psi}$ *be a family of monotone functions* $\bar{\psi} : D_I^{\rho(\psi)} \to D_{I_\perp}$. *Let* p *and* ν *be as in lemma (4.2.3) with* $D_{I_\perp}$ *instead of* D_I, *then* ν *satisfies all the equations of* Γ_p.

This theorem is analogous to theorem (4.2.2) and is stated with respect to the attribute grammar $G^{\Omega}_{\Sigma,I}$.

(4.3.4) Theorem :

Let $\psi \in \Psi_k$, $\bar{d} \in D_I^k$. *For all* d' *in* D_I ($d' \neq \perp$), $\psi_{I,\underline{\text{name}}}(\bar{d}) = d'$ *iff there exists a tree* t *in* $M(P_\Omega)_\psi$ *such that* $(\bar{d},d') \in R_{t,I_\perp}$.

Proof : Let $\psi_{I,\underline{\text{name}}}(\bar{d}) = d' \neq \perp$. There exists a call-by-name computation γ : $\psi(\bar{d}) \overset{n}{\to} d'$. We show the existence of t by induction on n, and for all $\bar{d}$ in $(D_I \cup \{\perp\})^k$ (and not only for $\bar{d}$ in D_I^k), and, of course, for all ψ in Ψ (and $k = \rho(\psi)$).

The computation γ is of the form :

$$\psi(\bar{d}) \to s_\psi[\bar{d}/\bar{x}] \overset{n-1}{\to} d'$$

By lemma (4.3.2) the last n-1 steps of γ can be written $\gamma';\gamma''$ (where γ' and γ'' are as in (4.3.2)). Hence γ'' is of the form $s_i[\bar{d}/\bar{x}] \overset{m}{\to} d'$ with $m < n$.

Let $p = p_{\psi,i}$ and $\alpha(p) = \psi_1 \ldots \psi_\ell$.

The tree t we are defining will be of the form $p(t_1,\ldots,t_\ell)$ where $t_1,\ldots,t_\ell$ are associated (by induction) to computations of $\psi_1(-)$, ..., $\psi_\ell(-)$ respectively. Let us precise this point.

Let us remark that for certain subterms s of s_i, one can "extract" from γ" a call-by-name computation $s[\bar{d}/\bar{x}] \overset{h}{\to} d_s$ of length $h \leq m$, with $d_s \in D_I$ ($d_s \neq \perp$). For others, this is impossible ; this means that their value is not needed for the evaluation of $s_i[\bar{d}/\bar{x}]$ (the call-by-name strategy allows such a situation). In the latter case we take $d_s = \perp$.

Consider now $w_1, w_2,\ldots, w_\ell$, the list of nodes of s_p where some symbol of Ψ occurs, so that ψ_j is the left-most symbol of $s'_j = s_i/w_j$.

If $d_{s'_j} = \perp$ then we set $t_j = \Omega_{\psi_j}$. Otherwise, let us consider $s'_j = \psi_j(r_1,\ldots,r_{k'})$ and let $\bar{d}'^j = (d'_1,\ldots,d'_{k'})$ where $d'_{j'} = d_{r_{j'}}$.

One can extract from γ" a call-by-name computation :

$$\psi_j(\bar{d}') \overset{m'}{\to} d_{s'_j} \qquad \text{with } m' \leq m.$$

One takes for t_j its associated tree (by induction since $m' \leq m < n$).

Hence we have defined $t = p(t_1,\ldots,t_\ell)$. We need only verify that $(\bar{d},\bar{d}') \in R_{t,I_\perp}$. Using induction again we only need to verify that Φ_p holds "at the root of t".

Lemma (4.3.2) shows that the Boolean condition B'_p holds.

That the equations of Γ_p hold is a consequence of lemma (4.3.3).

Conversely, it is easy to construct from t in $M(P_\Omega)_\psi$ such that $(\bar{d},\bar{d}') \in R_{t,I_\perp}$ a call-by-name computation $\psi(\bar{d}) \overset{*}{\to} d'$ to which the tree t corresponds by the first part of the proof. □

Since we have characterizations of $\psi_{I,\underline{val}}$ and $\psi_{I,\underline{name}}$ in terms of attribute grammars, the proof methods of section 2 apply and yield complete proof methods for the partial correctness of recursive applicative procedures.

This seems to be a new result for $\psi_{I,\underline{name}}$. The case of $\psi_{I,\underline{val}}$ is a consequence of Gallier's results via the transformation of remark (4.2.6).

(4.3.5) Remarks.

Here is an example showing that the computation of $\psi_{I,\underline{name}}(\bar{d})$ by means of $G^{\Omega}_{\Sigma,I}$ avoids certain duplications, hence is shorter than its evaluation by call-by-name rewriting sequences. The reason is that the evaluation by means of attributes corresponds to a certain sharing of duplicated subexpressions. Consequently, this construction fails for call-by-name computations of non deterministic schemes.

We note by passing that $G_{\Sigma,I}$ works for call-by-value computations of non-deterministic schemes (the construction of remark (4.2.6) also works in this case).

Let Σ be the following equation :

$$\psi(x,y) = \underline{\text{if}}\ qx\ \underline{\text{then}}\ f(y,y)\ \underline{\text{else}}\ \ \psi(hx,\psi(gx,hy))$$

In a call-by-name computation of the form

$$\psi(d,d') \to \psi(hd, \psi(gd,hd'))$$

$$\to f(\psi(gd,hd') , \psi(gd,hd'))$$

$$\to \ldots$$

the two occurences of $\psi(gd,hd')$ are computed separately.

In the computation by attributes one evaluates the tree shown on Fig.17 where t' corresponds to the evaluation of $\psi(gd,hd')$. This tree t' is not duplicated as was $\psi(gd,hd')$ in the call-by-name computation. In other words, the computation by attributes corresponds to an implementation of recursivity by sharing as in Vuillemin [Vui 75].

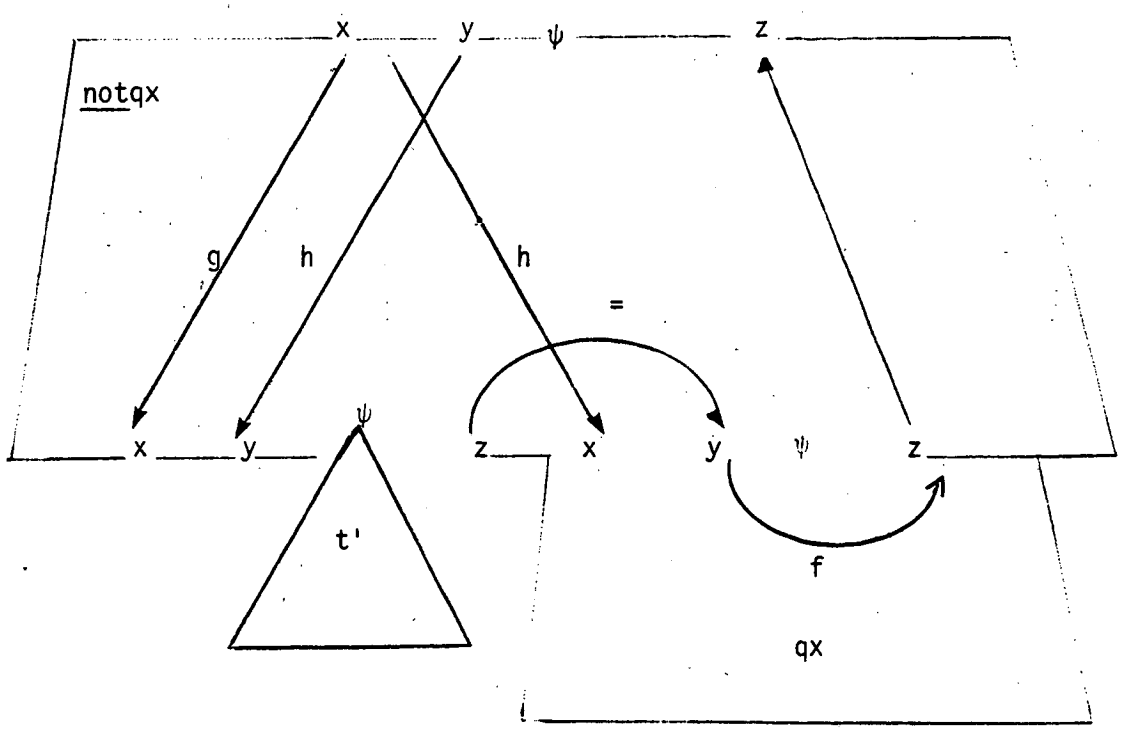


Fig. 17

If now the definition of $\psi(x,y)$ is augmented by some alternative cases making it non-deterministic, the two occurrences of $\psi(gd,hd')$ may be evaluated into two distinct values and this cannot be handled by the attribute grammar. □

(4.3.6) Remark :

We claim that the computation rule $\mathscr{C}$ of remark (4.1.5) and the computation rules dealing with parallel and and or (see definition (4.1.2) and Vuillemin [Vui75]) can be formalized with appropriate attribute systems.

In each case a complete inductive assertion proof method can be derived from the corresponding formalization. □

BIBLIOGRAPHY

[Cad 75] Cadiou J.M.
"Recursive Definition of partial Functions and their Computations"
Ph.D, Stanford. (1972)

[CF 82] Courcelle B., Franchi-Zannettacci P.
"Attribute Grammars and Recursive Program Schemes (I and II)"
Theoretical Computer Science 17, 2 and 3. 163-191 and 235-257 (1982)

[CM 79] Chirica L.M., Martin D.F.
"An Order-Algebraic Definition of Knuthian Semantics"
Mathematical Systems Theory 13, 1. 1-27 (1979)

[Cou 84] Courcelle B.
"Attribute Grammar: Definitions, Analysis of Dependencies, Prof Methods"
Methods and Tools for Compiler Construction CEC-INRIA Course (B.Lorho ed.), Cambridge University Press. (1984)

[Der 83] Deransart P.
"Logical Attribute Grammars"
IFIP'83, Paris (R.E.A. Mason ed.), North-Holland Publishing Co.. 463-469 (September 1983)

[Der 84] Deransart P.
"Validation des Grammaires d'Attributs"
thèse d'Etat, Université de Bordeaux I
(October 1984)

[DJL 83] Deransart P., Jourdan M., Lorho B.
"Speeding up Circularity Tests for Attribute Grammars"
rapport RR 211, INRIA, Rocquencourt (May 1983). Acta Informatica 21, 375-391 (1984)

[DM 75] De Bakker J.W., Meertens L.G.L.T.
"On the Completness of the inductive Assertion Method"
J. Comput. Syst. Science 11,3. 323-357 (1975)

[DPSS 77] Duske and al.
"IO-macrolanguages and Attributed Translations"
Information and Controle 35. 87-105 (1977)

[DS 76] Downey P., Sethi R.
"Correct Computation Rules for recursive Languages"
SIAM J. Comput. 5. (1976)

[EF 81] Engelfriet J., Filè G.
"The Formal Power of One-Visit Attribute Grammars"
Acta Informatica 16, 3. 275-302 (1981)

[EF 82a] Engelfriet J., Filè G.
"Simple Multi-Visit Attribute Grammars"
JCSS 24, 3. 283-314 (June 1982)

[EF 82b] Engelfriet J., Filè G.
"Passes, Sweeps, and Visits in Attribute Grammars"
memorandum INF-82-6, Onderafdeling der Informatica, Technische Hogeschool Twente. (August 1982)

[Eng 84] Engelfriet J.
"Attribute Grammars: Attribute Evaluation Methods"
Methods and Tools for Compiler Construction (B. Lorho ed.), INRIA-CEC Course, Cambridge University Press. 103-138 (1984)

[Gal 81] Gallier J.
"Nondeterministic Flowchart Programs with recursive Procedures: Semantics and Correctness"
TCS 13. 193-229 and 239-270 (1981)

[GTW 77] Goguen J.A., Thatcher J.W., Wagner E.G., Weight J.B.
"Initial Algebra Semantics and continuous Algebras"
JACM 24. 68-95 (January 1977)

[Gue 81] Guessarian I.
"Algebraic Semantics"
LNCS 99. (1981)

[KH 81] Katayama T., Hoshino Y.
"Verification of Attribute Grammars"
8th ACM POPL, Williamsburg, VA. 177-186 (January 1981)

[May 81] Mayoh B.H.
"Attribute Grammars and Mathematical Semantics"
SIAM Jour. on Computing 10, 3. 503-518 (August 1981)

[MC 68] Mc Carthy J. and al.
"LISP 1.5"
MIT Press. (1965)

[PAN 79] Pair C., Amirchahy M., Nèel D.
"Correctness Proofs of Syntax-directed Processing Descriptions by Attributes"
JCSS 19, 1. 1-17 (August 1979)

[Vui 74] Vuillemin J.
"Correct and optimal Implementations of Recursion in a simple programming Language"
JCSS 9. 1-17 (1974)

[Vui 75] Vuillemin J.
"Syntaxe, Sèmantique et Axiomatique d'un Langage de Programmation simple"
Doctoral Dissertation, Birkhauser Verlag. (1975)

[Wan 78] Wand
"A new Incompletness Result for Hoare's System"
JACM 25. 168-175 (1978)

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique